COMPARATIVE ANALYSIS AND DESIGN OF STREAM CIPHER

THESIS

SUBMITTED FOR THE AWARD OF THE DEGREE OF

Doctor of Philosophy

IN

COMPUTER SCIENCE

SUBMITTED BY:

FAHEEM SYEED MASOODI

UNDER THE SUPERVISION OF

DR. MOHAMMAD UBAIDULLAH BOKHARI

DEPARTMENT OF COMPUTER SCIENCE
ALIGARH MUSLIM UNIVERSITY
ALIGARH (INDIA)

2013

To My Family

For Love, Care and Support

CANDIDATE'S DECLARATION

I, **Faheem Syeed Masoodi, Department of Computer Science** certify that the work embodied in this Ph.D thesis is my own bonafide work carried out by me under the supervision of **Dr. Mohammad Ubaidullah Bokhari** at Aligarh Muslim University, Aligarh. The matter embodied in this Ph.D. thesis has not been submitted for the award of any other degree.

I declare that, I have faithfully acknowledged, given credit to and referred to the research workers wherever their works have been cited in the text and the body of the thesis. I further certify that I have not willfully lifted up some other's work, para, text, data, result, etc. reported in the journals, books, magazines, reports, dissertations, theses, etc., or available at web-sites and included them in this Ph.D. thesis and cited as my own work.

Date: 20-6-2013

(Faheem Syeed Masoodi)

CERTIFICATE FROM THE SUPERVISOR

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Signature of the Supervisor:.....

Name & Designation: Dr. Mohammad Ubaidullah Bokhari

(ASSOCIATE PROFESSOR)

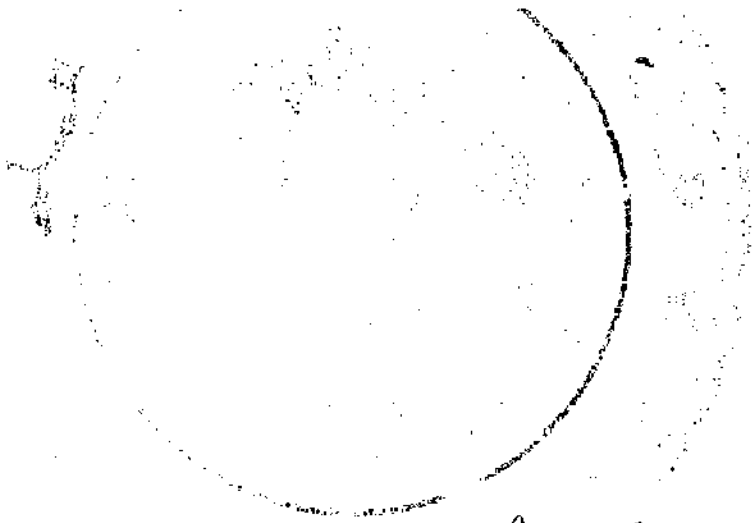
Department: COMPUTER SCIENCE

(Signature of the Chairman of the Department with seal)

CHAIRMAN
Department of Computer Science
A.M.U., Aligarh

COURSE/ COMPREHENSIVE EXAMINATION/ PRE-SUBMISSION SEMINAR COMPLETION CERTIFICATE

This is to certify that **Mr. Faheem Syeed Masoodi**, Department of Computer Science has satisfactory completed the course work/comprehensive examination and pre-submission seminar requirement which is part of his Ph.D programme.



Date: 20/6/2013

(Signature of the Chairman of the Department)

CHAIRMAN
Department of Computer Science
A.M.U., Aligarh

COPYRIGHT TRANSFER CERTIFICATE

Title of the Thesis: Comparative Analysis and Design of Stream Cipher

Candidate's Name: Faheem Syeed Masoodi

Copyright Transfer

The undersigned hereby assigns to the Aligarh Muslim University, Aligarh, copyright that may exist in and for the above thesis submitted for the award of Ph.D. degree.

(Faheem Syeed Masoodi)

ABSTRACT

Cryptographic primitives form the fundamental building blocks for cryptographic applications and protocols and one of the oldest cryptographic primitive is Symmetric-key cryptography, in which an identical key is used for both encryption and decryption of confidential data, i.e. $K_e = K_d = K$, where e and d are encryption and decryption algorithms respectively and K is the shared key. On the basis of high level structure in which they encrypt data, symmetric-key cryptography is broadly classified as block ciphers and stream ciphers.

A stream cipher provides a method for obtaining confidentiality of data: only the party with knowledge of the secret key can obtain the content. The research in this thesis is a contribution to the on-going research on the analysis and design of stream ciphers. Several different existing stream ciphers are analysed and a new stream cipher called "BOKHARI Stream Cipher" is presented.

This thesis begins with an introduction to theoretical foundations of modern day cryptography, followed by a discussion on cryptographic primitives. Since this thesis is devoted to stream ciphers, the symmetric key cryptography is analysed in detail and a study on various modes of operation in which symmetric key ciphers operate is presented. Towards the end of chapter, a discussion is carried out on cryptanalytic attacks on cryptographic protocols with the thrust being upon the attacks related to stream ciphers.

Next, the very concept of stream cipher and different design approaches of stream cipher development are discussed. Linear feedback shift registers (LFSR) as maximal length sequence generators are widely used in stream ciphers for key stream generation due to their good statistical properties, large period, low implementation costs and are readily analysed using algebraic techniques. LFSRs alone are notoriously insecure from a cryptographic standpoint due to their inherent linearity.

Cryptographically strong pseudo-random sequences are produced by combining more than one LFSR with methods like Non-linear combination generator, Non-linear filter generator and Clock-controlled generator in order to introduce non linearity.

Following this, an analysis of several existing stream ciphers including SOBER family of stream ciphers, Grain-128, NLSv2 , DRAGON stream cipher, PY (ROO) family of stream ciphers is presented. An attempt is made to assess the reliability of these ciphers in terms of security, memory requirements, speed and implementation.

The final part of this thesis presents a new stream cipher called "BOKHARI Stream Cipher", a new proposed software-oriented stream cipher constructed using a Non-linear feedback shift register and a Nonlinear filter function and is designed for a secret key that is up to 128 bits in length. The BOKHARI stream cipher uses basic design principles from the stream cipher SOBER-t16 and transformations derived from the steam cipher DRAGON. Security and Statistical analysis of BOKHARI stream cipher is performed, which shows that no bias was found by any of 15 tests included in National Institute of Standards and Technology (NIST) suite and six tests of ENT test suite.

ACKNOWLEDGEMENT

ALLAH, the Almighty, the lord of all worlds, the most merciful, the most gracious, be witness, I acknowledge you every minute and greatest blessing on me, apparent and hidden favours, from the time before, to the time after this life, in all the quarters, including my research and writing of this PhD thesis, just for my belief in you.

At the beginning of my doctoral research, I could barely foresee how much I was to mature and gain intellectually in the years to come. A lot many people are responsible for this development; unfortunately, I am able to thank only a few here.

I avail this unique opportunity to express my gratitude and indebtedness to my supervisor Dr. Mohammad Ubaidullah Bokhari. It is not justified if I simply thank him for steering me through the journey that has culminated in this thesis, giving me ample freedom to pursue my topics of interest, patiently reviewing my papers, providing timely and insightful comments (often at short notice), and very generously allowing me to travel to a number of workshops and conferences. I have come to realise that the best way to express my gratitude for the support provided by him and for the role model he has been, is through continued contributions to cryptology – I am hopeful of doing so. Besides my advisor, I owe a great debt of gratitude to Dr. Mehreen Afzal of National University of Science and Technology, Pakistan for the invaluable suggestions and guidance during my Ph.D, Thank you ma'am. I am also very thankful to Prof. Ricardo Dahab for inviting me to *SP*-Crypto 2011, Brazil and giving me an opportunity to sit among reputed cryptographers and boost my morale. Needless to say, I look forward to many more such fruitful collaborations. I would like to thank the anonymous referees of my papers for their constructive comments.

I am highly grateful to Department of Computer Science, Aligarh Muslim University for providing all necessary facilities for the successful completion of my research. A special thanks to Dr. Rafiq-ul-Zaman khan for his help during my course work. I would like to thank UGC-MANF for taking care of my financial requirements throughout these years.

As a Ph.D student, I spent most of my time in my lab. I would like to thank my fellow lab mates: Shadab, Sadaf, Shamas, Hatem and Faraz. You have all been great friends. Besides research activities, I also enjoyed my stay at residential Hostel and this time wouldn't have been fun, had it not been with people like Dr Manzoor, Jameel, Mohsin, Nadeem, Javeed, Amin, Ayaz, Iqbal, Shadab and Irshad. It would never been same, had it not been the illogical arguments of Javeed in every form of discussion, which were equally torture at times. Thanks mate for everything.

I am highly obliged to my cousin brother Athar Masoodi, who was always a morale booster and indefinite support at hard times during my stay at Aligarh.

It is a pleasure to thank many of my friends back home (especially those from my undergrad years) for their constant encouragement. Again, since there are too many of them, I am sorry at my inability to provide names here. A special mention of Sajad, Waseem and Adil though is called for.

This work would not have been possible without my experiments. I acknowledge my hp pavilion PC for running my experiments and drafting my papers and this thesis. I also acknowledge the suffering of ENTER key. You did well.

I would like to thank Iram for always being understanding and best possible sister and Gousiya for always encouraging me when things get tough. You have been the best friend.

I am at a loss for words to thank my family for being the best people, I know. No words are ever sufficient to express my everlasting love, gratitude, appreciation and thanks to my Dad, Mom, Bayaji, Bhairaj and Baya. I hope my three B's will enjoy reading this thesis.

(Faheem Syeed Masoodi)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS & SYMBOLS

AES	Advanced Encryption Standard
DES	Data Encryption Standard
ECRYPT	European Network of Excellence in Cryptology
NESSIE	New European Schemes for Signatures, Integrity and Encryption
NIST	National Institute of Standards and Technology
MAC	Message Authentication Code
OWF	One Way Function
WCR	Weak Collision Resistance
SCR	Strong Collision Resistance
MD	Message Digest
SHA	Secure Hash Algorithm
TWOF	Trapdoor One-Way Function
NSA	National Security Agency
PKC	Public Key Cryptography
RSA	Rivest-Shamir-Adleman
ECB	Electronic Code Book
CBC	Cipher Block Chaining
CFB	Cipher Feedback
OFB	Output Feedback
CTR	Counter
PRG	Pseudo Random Generator
CRYPTREC	Cryptography Research and Evaluation Committee
PRBG	Pseudo Random Bit Generators
CSPRBG	Cryptographically Secure Pseudo Random Bit Generator
IV	Initialization Vector
OTP	One Time Pad
MOC	Maximum Order Complexity

FSR	Feedback Shift Register
LFSR	Linear Feedback Shift Register
NFSR	Nonlinear Feedback Shift Register
FCSR	Feedback Shift Register with Carry
S-Box	Substitution Box
ANF	Algebraic Normal Form
SCA	Side Channel Analysis
SSCA	Simple Side Channel Analysis
DSCA	Differential Side Channel Analysis
SPA	Simple Power Analysis
DPA	Differential Power Analysis
NLF	Nonlinear Function
FPDS	Full Positive Difference Set
XOR	Exclusive-Or
RFC	Request for Comment

SYMBOLS

$\in$	Belongs to
$\oplus$	XOR
$\mathbb{F}_q$	Finite Field with q elements
$\mathbb{F}_2$	Binary Field
GF(2)	Galois Field with two elements 0 and 1
$\sum$	Integer Addition
$\mathbb{Z}_2$	Integer Field with values 0 and 1
$\boxplus$	Addition Modulo 2^w
Θ	Modular Multiplication
mod2	Modulo 2

Introduction

Chapter 1

Introduction

Information is an asset and security is ubiquitous. The motivation to keep information secure is obvious in many circumstances such as e-commerce, communications and electronic transactions. The various objectives of Information security are authentication, authorization, validation, non-repudiation, availability and confidentiality. The need to protect information goes back to antiquity. Since the early days of writing, heads of state and military commanders understood that it was necessary to provide some mechanism to protect the confidentiality of written correspondence and to have some means of detecting tampering. There are several popular examples of secret writing. Julius Caesar, a Roman military and political officer in first century BC is credited with the invention of Caesar cipher or shift cipher. In Caesar cipher, each letter in the alphabet was replaced by letter three positions later. One more such popular example is the story of Histiaeus. His solution of secrecy was that he shaved the head of his messenger, stained the message on his head and allowed the hair to grow back to cover the stained message before he sent him away.

The word Cryptology has its origin in Greek words *kryptos*, which means hidden and *logos*, which means word. It is believed to have emerged from ancient Egypt more

than 4500 years ago. It has predominantly been an art form used by the military and governments for message confidentiality. The transformation of cryptology from an art to science may have been motivated by the popularity of wireless communication in 1920's. However, it is the birth of computer era that arguably made open research in cryptography possible. Cryptology witnessed a radical change with the advent of computer and is no more considered as a science of secret writing. It has gone from an art form that dealt with secret communications for military to a science that helps to secure system for ordinary people all across the globe. In addition to the computer science, cryptology uses ideas from several other fields such as information theory, number theory, abstract algebra and quantum theory. Modern cryptology encompasses *message authentication, digital signatures, and protocols for exchanging secret keys, authentication protocols, electronic auction, election, elections and digital cash.*

Following is a glossary of some fundamental terms.

Encryption is the process of converting a message (*or plaintext*) into an 'unintelligible' form (called *ciphertext*) and *decryption* is the reverse process. The ciphertext is transmitted by the sender to the intended recipient of the plaintext, across an insecure communication channel (any third party can intercept data that flows through such a channel). The algorithm used for performing encryption and decryption is called the *cipher*. A *cryptosystem* can mean any system that involves cryptology or just the algorithms, along with sets of possible inputs to them, that are used to perform encryption and decryption of messages.

Cryptology, the study of cryptosystems, can be subdivided into two disciplines. Cryptography concerns itself with the design of cryptosystems, while cryptanalysis studies the breaking of cryptosystems. These two aspects are closely related; when setting up a cryptosystem the analysis of its security plays an important role. Cryptography is broadly classified on the basis of number of keys used for encryption/decryption. Generally cryptography is classified as unkeyed cryptography, Asymmetric key cryptography and Symmetric key Cryptography. In this thesis, the primary focus is Stream ciphers, which is a variation of Symmetric key cryptography. In symmetric key cryptography, a single shared key is used for both encryption and

decryption and the key needs to be kept secret. Symmetric ciphers on the basis of the size of block which they encrypt, falls under two categories: block ciphers and stream ciphers. Block ciphers are stateless and given plaintext directly output ciphertext. Stream ciphers maintain state and produce keystream, which is externally combined with the plaintext to make the ciphertext. Stream cipher takes as input the secret key and a parameter called initialization vector (IV) and generates a stream of bits called the keystream, which is XORed with plaintext to produce ciphertext. On the basis of the internal state they maintain, stream ciphers are commonly classified as synchronous and asynchronous stream ciphers. Synchronous stream cipher generates the keystream independent of the plaintext and the ciphertext while as Asynchronous stream cipher generates the keystream based on the previous ciphertext. Stream ciphers have played and continue to play an important role in cryptography.

This thesis deals with techniques of stream cipher design that allow efficient implementation in conjunction with high security. Much of the doctoral period has been spent in evaluating algorithms submitted to the open competition eSTREAM [ECR10] and NESSIE [NES12]. eSTREAM was a multi-year effort with the explicit goal to advance state-of-the-art knowledge about stream cipher design. Sponsored by the European Network of Excellence in Cryptngraphy (ECRYPT), the project began in 2004 with proposals for new stream ciphers being invited from industry and academia. These proposals were intended to satisfy either a software-oriented or a hardware-oriented profile (or both if possible). The original call for proposals generated considerable interest with 34 proposals being submitted to the two different performance profiles [MRO08]. The project ended in April 2008 with an announcement of a portfolio of eight stream ciphers, four in each profile. This portfolio was revised in September 2008 when one hardware-oriented cipher, F-FCSR-H v2, was eliminated.

In 2000 a project called New European Schemes for Signatures, Integrity, and Encryption (NESSIE) was initialized. The aim of this project was to collect a strong portfolio of cryptographic primitives. After an open call for proposals the submissions were thoroughly evaluated.

Aim of thesis

In this work, performance aspects of various stream ciphers like SOBER family of stream ciphers [MAB11], PY family of stream ciphers [BAM10] and Dragon stream cipher [CHE04] is evaluated. The aim of this thesis is to propose a new 128-bit software oriented stream cipher called BOKHARI stream cipher [BM12].

Thesis Outline

This thesis, as its name suggests, is devoted to the area of stream ciphers. It will show new ideas in both stream cipher analysis and stream cipher design. The rest of the thesis is outlined as follows.

Chapter 2 introduces the reader to the theoretical foundations of modern day cryptography, followed by a discussion on cryptographic primitives. Since this thesis is devoted to stream ciphers, this thesis analyses the symmetric key cryptography in detail and present a study on various modes of operation in which symmetric key ciphers operate. Towards the end of chapter, a discussion is carried out on cryptanalytic attacks on cryptographic protocols with the thrust being upon the attacks related to stream ciphers.

Chapter 3 discusses the very concept of stream cipher and different design approaches of stream cipher development. Different pseudorandom generation techniques are discussed with the focus being on linear feedback shift register (LFSR). Linear feedback shift registers as maximal length sequence generators are widely used in stream ciphers for key stream generation due to their good statistical properties, large period, low implementation costs and are readily analysed using algebraic techniques. Then this thesis discusses cryptographically strong pseudo-random sequences that are produced by combining more than one LFSR with methods like Non-linear combination generator, Non-linear filter generator and Clock-controlled generator in order to introduce non linearity.

Chapter 4 presents an analysis of several existing stream ciphers. This chapter first analyses the SOBER stream cipher and its respective variants, which have been developed to strengthen its security or to be suitable for 16-bit and 32-bit processors.

Nearly all SOBER family ciphers consists of three basic components, there is a Linear Feed-back Shift Register (LFSR), which uses a recursion formula to generate the stream of pseudo-random bits. Next a Non-Linear Function (NLF) combines these words in a non-linear way to produce the NLF-stream. Finally, the so-called stuttering produces the keystream by decimating the NLF-stream in an irregular fashion. A comparative study of various variants of SOBER was carried out in terms of key loading, byte ordering, memory requirements, polynomials employed and implementation of LFSR.

The members of Py family are software-oriented synchronous stream ciphers that are based on arrays, modular additions and bit-rotations. Py was one of the candidates in eSTREAM profile 1 stream cipher category but was found to be susceptible to some cryptanalytic attacks. Several improved versions of Py were submitted in form of Pypy, TPy, TPy6 and TPypy. This chapter analyzes the design and security specifications of Py cipher and its variants to assess the reliability of these ciphers and get a better understanding for the design of a stream cipher that is more secure as well as efficient. Finally chapter four discusses some Non-Linear feedback shift register based stream ciphers including DRAGON, GRAIN-128 and $NLSV_2$ The DRAGON stream cipher was submitted to the eSTREAM project in 2005 and had advanced to Phase 3 of the software profile. The design of the Dragon stream cipher incorporates a single word-based non-linear feedback shift register and a non-linear filter function with memory. The cipher uses the minimum number of operations to ensure its security, yet maintains a high level of efficiency. With a keystream throughput of 6.7 cycles per byte and a key generation setup time of 1,395 cycles, Dragon is competitive with other Modern word based stream ciphers. GRAIN-128 is a bit oriented synchronous stream cipher. It uses one NFSR of size 128 and one LFSR of size 128 with a filter function to combine the output of these shift registers.

Chapter 5 of this thesis introduces a new stream cipher called "Bokhari stream cipher", a new proposed software-oriented stream cipher constructed using a Non-linear feedback shift register and a Non linear filter function and is designed for a secret key that is up to 128 bits in length. The BOKHARI stream cipher uses basic design principles from the stream cipher SOBER-t16 and transformations derived from the steam cipher DRAGON. At the end of this chapter, security and statistical

analysis of BOKHARI stream cipher is performed, which shows no bias was found by any of 15 tests included in NIST suite and six tests of ENT suite.

Chapter 6 summarizes the contributions of the thesis and gives concluding remarks.

Appendix I contains list of papers published during Ph.D.

Appendix II contains the source code of BOKHARI stream cipher.

Cryptology

Chapter 2

Cryptology

The word 'Cryptology' has a Greek etymology and stems from word *kryptos*, which means hidden and *logos*, which means study. Cryptology, the study of cryptosystems, can be subdivided into two disciplines-Cryptography and Cryptanalysis. Cryptography concerns itself with the design of cryptosystems, while cryptanalysis studies the breaking of cryptosystems. These two aspects are closely related: when setting up a cryptosystem the analysis of its security plays an important role.

Cryptography is the discipline of encrypting a message in an incomprehensible form with the use of some frequently changing key text, and decryption of this incomprehensible text back into the actual message, with the aim of protecting a secret from adversaries, intruders, interlopers, eavesdroppers or simply attackers. According to RFC2828 [SHH13], cryptography can be termed as "The mathematical

science that deals with transforming data to render its meaning unintelligible (i.e., to hide its semantic content), prevent its undetected alteration, or prevent its unauthorized use. If the transformation is reversible, cryptography also deals with restoring encrypted data to intelligible form." The transformation of *plaintext* (actual message) to *ciphertext* (incomprehensible form) in order to keep the content hidden from unintended parties is known as *encryption* and the reverse process of transforming the cipher text into plaintext is called *decryption*. The need encryption and decryption comes into play because of the insecure channel over which the sender transmits the message to the intended recipient. The aim of cryptography is not to hide the existence of message, but to rather hide its meaning [SIM00]. Cryptography studies the design of algorithms and protocols for information security.

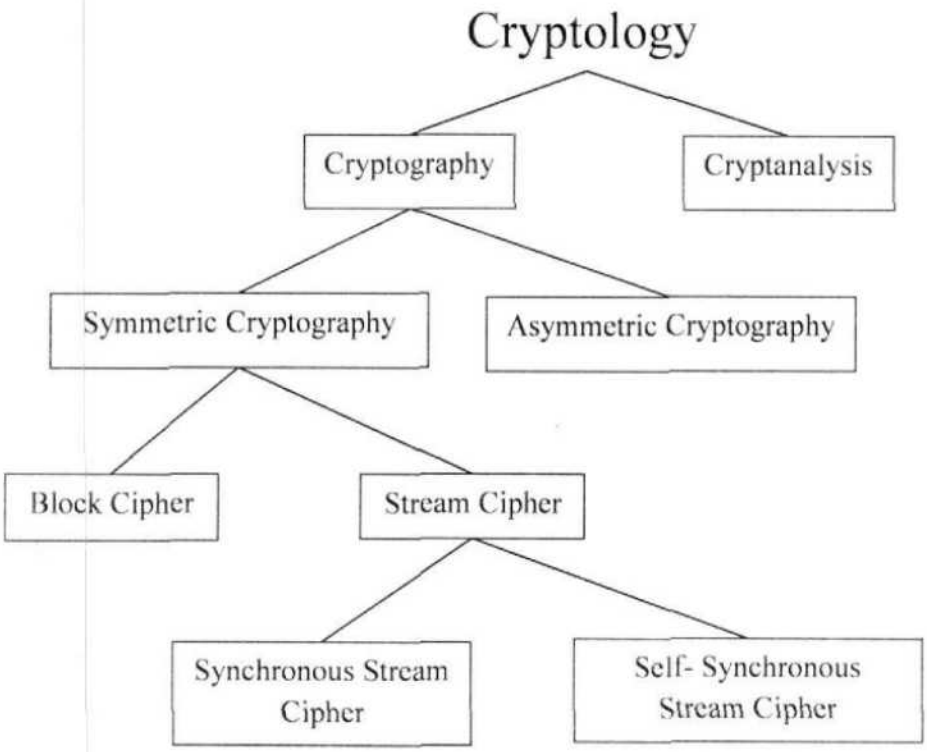


Fig 1: Classification of Cryptology

The need to keep information secret especially in communications is obvious in many circumstances such as military, diplomatic and business affairs. Following are the different security services of cryptography:

- ***Data Confidentiality***. The aim is to assure that the information is available only to authorized recipients. Confidentiality of data used to be the original purpose of cryptography and it is accomplished by using strong encryption algorithms.
- ***Data Integrity***. This service guarantees that the content of the message, that was sent, has not been tampered with. The unauthorized alteration of data must be detected and only authorized users are allowed to alter the data. Data alteration includes such things as insertion, deletion, substitution and replication. Generally techniques like *message authentication codes* (MACs) and *digital signatures* are used to protect integrity of data.
- ***Authentication*** is the act of confirming identity of an entity and establishing whether someone or something is what they claim to be. Besides entity identification, authentication also aims at data origin authentication. Data origin authentication of information confirms the origin, content and time of creation. Authentication is generally accomplished by using passwords.
- ***Non-repudiation*** The goal is to ensure that either of the parties involved cannnt deny any previous commitments or contracts. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, if either of the parties has signed a contract, neither of them should be able to claim that it was someone else who signed the contract and them. Non-repudiation can be achieved by using digital signatures together with digital certificates.

2.1 Cryptographic Primitives

The cryptographic objectives discussed above are addressed using fundamental tools or algorithms commonly known as *Cryptographic primitives*. These primitives are classified into three groups. *Unkeyed primitives*, *Symmetric-key primitives* and *Asymmetric-key primitives*. The focus of this thesis is on symmetric key cryptography, particularly *stream ciphers*.

2.1.1 Unkeyed Cryptography

Unkeyed primitives play a vital role in cryptographic services mainly *authentication* and *data integrity*. The name unkeyed primitives originates from the fact that they do

not use secret keys; therefore no secrecy is involved in their algorithms. The family of unkeyed primitives consists mainly of tools like *hash functions* and *one way permutations*. The term *Hash function* has its roots in computer science, where it denotes a function that compresses a string of arbitrary length (virtual address) to a string of fixed length (physical address). A hash function is a computationally efficient function that takes strings of arbitrary length as input and map these to short fixed length output strings, commonly known as *message digest or checksum*. Hash functions are fundamental components of many cryptographic applications such as digital signatures, integrity protection and random generation *etc*. Desirable properties of a good hash function are *uniformity*, *weak collision resistance (WCR)*, *strong collision resistance (SCR)* and *ease of computation*. Some of the widely used hash functions used are MD5[KR95], SHA-2(SHA-224, SHA-256, SHA-384, SHA-512)[NIS12]. MD5 is now considered to be cryptographically broken and several attacks were reported on SHA-2 family. In an effort to replace older SHA-1 and SHA-2, **USNIST** on Nov. 2, 2007 announced a public competition to develop a new cryptographic hash algorithm that will be called as SHA-3. NIST selected five SHA-3 finalists - ***BLAKE, Grøstl, JH, Keccak,*** and ***Skein*** to advance to the third (and final) round of the competition on December 9, 2010 and on March 22-23, 2012 the final round conference was held in Washington D.C to discuss the SHA-3 finalist algorithms, and to solicit public feedback before NIST selects a winning algorithm for standardization. On October 2, 2012 NIST announced ***Keccak*** as the winner of its five-year competition. *Keccak* (pronounced "catch-ack"), was created by Guido Bertoni, Joan Daemen and Gilles Van Assche of STMicroelectronics and Michaël Peeters of NXP Semiconductors.

Properties of Ideal Cryptographic Hash Functions, It is

1. easy to compute the hash value for any given message,

2. Infeasible to find a message that has a given hash,

3. Infeasible to modify a message without hash being changed,

4. Infeasible to find two different messages with the same hash.

Message Authentication Codes (MACs) is a variation of hash functions that makes use of secret key and provide data origin authentication and data integrity. MACs can reduce the authenticity of a large quantity of information to the secrecy and authenticity of a short secret key.

2.1.2 Asymmetric Primitives

Asymmetric or public-key cryptography is based on the principle of making the encryption key effectively public knowledge, and hiding the details about the decryption key from unauthorized users. An asymmetric primitive defines unbalanced capabilities between sender and receiver [CAM10]. In 1976, Diffie and Hellman introduced the concept of public key cryptography [DH76] in order to overcome the limitation of communicating the secret key faced in symmetric key cryptography. Asymmetric cryptography algorithms make use of two different keys, a public key K_e used for encryption and a private key K_d used for decryption, i.e. $K_e \neq K_d$, where e and $d$are encryption and decryption algorithms respectively.

Public-key cryptography provides a nice way to help with the key management problem .Although it is computationally easy for the intended recipient to generate the public and private keys, to decrypt the message using the private key, and easy for the sender to encrypt the message using the public key but it is virtually impossible to deduce the private key based only on the knowledge of the public key. This is why, unlike symmetric key algorithms, a public key algorithm does not require a secure initial exchange of one (or more) secret keys between the sender and receiver. Asymmetric-key encryption (public-key encryption) provides confidentiality of data [CAM10].

2.1.3 Symmetric Primitives

Symmetric key cryptography also referred to as conventional or secret key cryptography derives its name from the fact that in this class of cryptographic primitive, encryption and decryption processes involve same (symmetric) key. Symmetric key cryptography was the only type of encryption in use prior to the development of public key cryptography in 1976. Symmetric cryptography algorithms make use of a single key for both encryption and decryption, i.e. $K_e = K_d = K$, where e

and d are encryption and decryption algorithms respectively and K is the shared key. The knowledge of encryption key is implicitly or explicitly equivalent to knowing the decryption key. These algorithms are typically fast, much more efficient than asymmetric algorithms in both hardware and software and are suitable for processing large streams of data. Symmetric cryptography also provides a degree of authentication because data encrypted with one symmetric key cipher cannot be decrypted with any other symmetric key cipher. Therefore, as long as the symmetric key cipher is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.

The disadvantage of symmetric key cryptography is that since the same key is used for both encryption and decryption therefore the key must be communicated in a secure manner prior to communication and known at both sender and receiver locations.

In the design of symmetric encryption systems, permutations and substitutions are usually used and combined to provide confusion and diffusion. The properties of confusion and diffusion were introduced by Shannon in his classic work "Communication Theory of Secrecy Systems" [SHA49] and defined as follows:

- ***Confusion***: "The method of confusion is to make the relation between the simple statistics of ciphertext and the simple description of plaintext a very complex and involved one", implies that the relationship between ciphertext and plaintext should be made so complex and obscure that the relationship can be determined between them.
- ***Diffusion***: "In the method of diffusion the statistical structure of which leads to its redundancy is *dissipated* into long range statistics", i.e. the influence of a single plaintext digit should be spread over a long range of ciphertext digits to hide the plaintext's statistical structure.

Massey in [MAS92] further interpreted confusion and diffusion as, by confusion "in a cipher, Shannon meant that the enciphering process should be such that the ciphertext statistics depend on the plaintext statistics in a manner too complicated to be exploited

by an attacker. By diffusion in a cipher, Shannon meant that each bit of the input key should influence many Bits of the ciphertext."

To summarize the above definitions, Confusion means that the key does not relate in a simple way to the ciphertext. In particular, each character of the ciphertext should depend on several parts of the key. Diffusion means that if a character of the plaintext is changed, then several characters of the ciphertext should change, and similarly, if a character of the ciphertext is changed, then several characters of the plaintext should change.

The combination of these two basic functions in multiple rounds is used to provide a maximum level of confusion and diffusion and good level of secrecy.

2.2 Classification of Symmetric Primitives:

Symmetric cryptography is broadly classified as *block cipher* and *stream cipher*. The names arise from the high level structure in which they encrypt data.

2.2.1 Block Cipher

Block cipher, one of the popular method of concealing information, works on the principle of dividing the input stream of plaintext into fixed size strings called blocks, typically 64, 128 or 256 bits long. In an attempt to generate all the blocks of same size, extra characters are generally added on to the end of the input stream, so that the total number of characters in the plaintext is exactly divisible by the block size. This procedure of appending additional data at the end of plaintext is called as Padding. The resultant blocks are then encrypted one at a time, using a cryptographic key to produce the ciphertext and there is no correlation between the encrypting of one message block and another.[ERI11] Consequently a block cipher maps plaintext message blocks of a specific length into ciphertext blocks of the same length. In practice, the vast majority of block ciphers either have a block length of 128 bits (16 bytes) such as the advanced encryption standard (AES), or a block length of 64 bits (8 bytes) such as the data encryption standard (DES) or triple DES (3DES) algorithm.

Block Cipher Modes nf operation:

A cryptographic mode usually combines the basic cipher, some sort of feedback, and some simple operations. The operations are simple because the security is a function of the underlying cipher and not the mode. Even more strongly, the cipher mode should not compromise the security of the underlying algorithm. The block cipher is typically used in one of the following five modes to encrypt plaintext blocks in order to break the property of the same plain text block always encrypting to the same cipher text block. All of the five modes have one goal: They encrypt data and thus provide confidentiality for a message exchanged between sender and receiver.

2.2.1.1 Electronic Code Book (ECB): This is the most classical scenario in which plaintext blocks are encrypted independently of one another with a secret key by the sending device. The resulting output blocks are concatenated to form the ciphertext and are transmitted to the decrypting device, where it is decrypted with the same key.

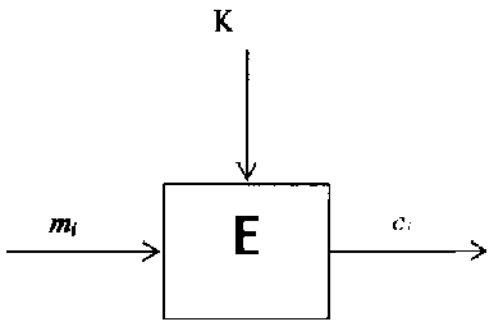


Fig 2.1: ECB mode encryption

One of the advantages of using ECB mode of encryption is that block synchronization between the sender and the receiver is not necessary, as the decryption of received encrypted blocks is still possible even if all encrypted blocks are not received due to transmission problems. Also, bit errors, e.g., caused by noisy transmission lines, only affect the corresponding block but not succeeding blocks and finally encryption with ECB is parallelisable.

The major limitation of the ECB mode is that it is highly deterministic. This means that identical plaintext blocks encrypted with same key result in identical ciphertext blocks.

2.2.1.2 Cipher block Chaining Mode (CBC): This mode of operation was designed to remove some of the disadvantages of the ECB mode. The main idea behind the working of Cipher Block Chaining (CBC) mode is that the output is a sequence of n-bit cipher blocks which are chained together so that each cipher block is dependent, not only on plaintext block from which it immediately came but on all previous plaintext blocks and also the encryption is randomized by using an initialization vector (IV).

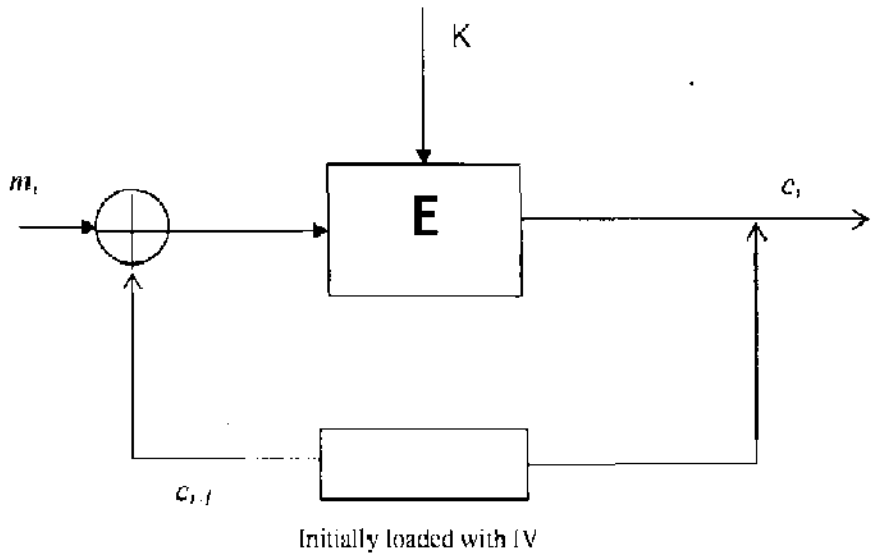


Fig 2.2: CBC mode encryptinn

2.2.1.3 Cipher Feedback Mode (CFB):

The cipher feedback (CFB) mode of operation uses the block cipher to generate a sequence of pseudorandom bits, and these bits are then XORed with plaintext bits to produce the ciphertext bits. The input shift register is initially filled with a seed value known as an initialization vector IV, and the underlying encryption algorithm is run once to produce output bits. The left-most s bits of the output bits are then XORed with the plaintext to generate the ciphertext. The result of this XOR operation is sent

over the network and also fed back to the input shift register, shifting the left-most s bits out. Then, the algorithm is run again and the next character is encrypted in the same manner. The IV need not be secret, since in the system it is in the position of a ciphertext but should be different for every message transmitted with the same key.

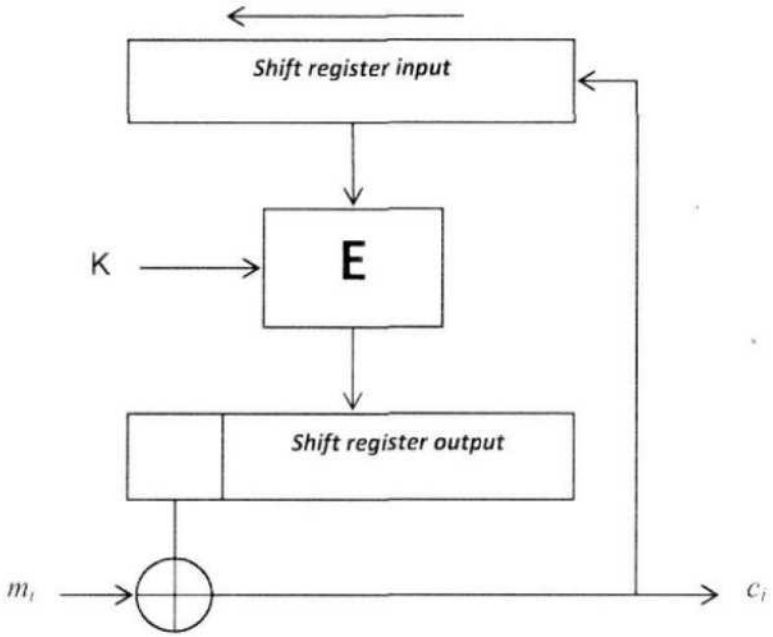


Fig 2.3: CFB mode encryption

The major advantage of the CFB mode is in applications that normally don't require the transmission of large messages. Since in CFB mode, block cipher emulates a stream cipher. Consequently, it is possible to encrypt blocks that are smaller than the block length of the block cipher. The limitation of CFB mode is that it suffers error propagation. Bit errors in the incoming cipher block will cause bit error at the same bit positions in the first plain text block and an incorrectly transmitted ciphertext block disturbs the decryption process until it "falls out" of the input register.

2.2.1.4 Output Feedback Mode (OFB):

The output feedback (OFB) mode of operation features feeding the successive output blocks from the underlying block cipher back to it. The output blocks form a string of bits which are XORed with the plaintext to generate the ciphertext. In contrast to CFB mode, the keystream generated in OFB mode is computed independent of both the plaintext and ciphertext. The OFB mode requires an IV as the initial random n-bit

input block, which is encrypted with the block cipher to generate first set of keystream bits. The following keystream bits as shown in Fig 4 are computed by feeding the previous cipher output back into the block cipher and encrypting it. The IV need not be secret, since in the system it is in the position of a ciphertext but should be different for every message transmitted with the same key.

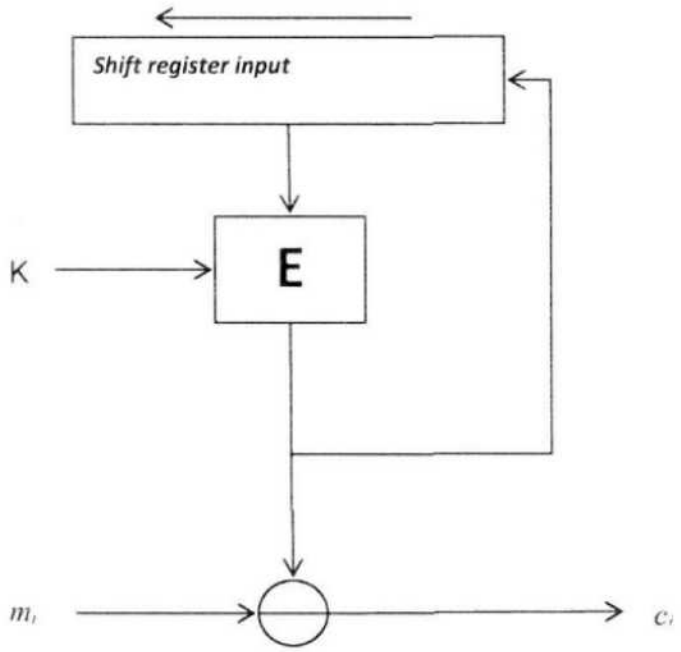


Fig 2.4: OFB mode encryption

OFB mode has an advantage over CFB mode in that any bit errors that might occur during transmission are not propagated to affect the decryption of subsequent blocks, as there are no chaining dependencies. A problem with output feedback is that by changing the ciphertext, the plaintext can be easily manipulated, but using a digital signature scheme can overcome this problem.

2.2.1.5 The counter (CTR) mode:

The counter (CTR) mode features feeding the underlying block cipher algorithm with a counter value which counts up from an initial value. Keystream block is generated by encrypting successive values of a "counter", which is then XORed with the plaintext blocks to generate the ciphertext. The counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual increment-by-one counter is the simplest and most popular. CTR mode

removes the limitations like error propagation and chaining dependencies faced in earlier discussed modes.

2.2.2 Stream ciphers:

Symmetric encryption system processes plaintext messages unit by unit, and as mentioned above, in the case of a block cipher a unit is called a block. Stream ciphers typically operate on individual units of plaintext. It takes as input the secret key and a parameter called initialization vector (IV) and generates a stream of bits called the keystream, which is XORed with plaintext to produce ciphertext. Stream ciphers have an internal state and on the basis of this state, stream ciphers are commonly classified as synchronous and asynchronous stream ciphers. Stream ciphers have played and continue to play an important role in cryptography.

Design and Analysis of Stream cipher is the main topic of this thesis and henceforth will be discussed more thoroughly in later chapters. For a general description of stream ciphers, see chapter 3.

2.3 Cryptanalysis:

Cryptanalysis is the technique of deriving the original message from the ciphertext without any prior knowledge of secret key or derivation of key from the ciphertext. According to RFC2828 [SHI13] the term cryptanalysis is used to refer to the "mathematical science that deals with analysis of a cryptographic system in order to gain knowledge needed to break or circumvent the protection that the system is designed to provide" A general technique for cryptanalysis, applicable to all cryptographic algorithms is to try all the possible keys until the correct key is matched, it is known as exhaustive key search. With every passing day, the computing ability of hardware is increasing manifold; therefore it becomes necessary to use long keys in order to avoid exhaustive key search. All the other attacks applied on stream ciphers are compared to exhaustive key search in terms of data and memory complexity and if its complexity is less than exhaustive key search, then only these are considered as successful. A symmetric key cipher, especially a stream cipher is assumed to be secure, if the computational capability required for breaking the cipher by best known attack is greater than or equal to exhaustive key search

2.3.1 Cryptanalysis Techniques

In this section, some generic attacks on stream ciphers are introduced. Most of the cryptanalysis on stream ciphers is performed under a known plaintext assumption. When comparing attacks, there are three complexity measures that are of interest.

Time complexity

Time complexity of an attack can be defined in terms of number of operations required to carry out that particular attack

Data complexity

Data complexity of an attack can be defined in terms of amount of observed keystream required to carry out an attack successfully

Memory complexity

Memory complexity of an attack can be defined in terms of amount of memory required to perform the attack successfully.

Some of the generic attacks on stream ciphers found in literature are discussed below.

2.3.1.1 Exhaustive Key Search Attack:

In an exhaustive key search attack or a brute force attack, the cryptanalyst tries all possible keys to decrypt a ciphertext and can be used against any cryptographic algorithm including stream ciphers except provable secure ciphers [PP10] though a provable secure cipher is not practically feasible, for e.g. one time pad. If the key size is n bits, then the attacker has to try on an average 2^{n-1} keys for breaking a cipher and 2^n keys in the worst case. The computational complexity of an attack is often stated as $O(2^n)$. [PP10] An attack with a higher computational complexity than an exhaustive key search is not considered an attack at all.

2.3.1.2 Side Channel Analysis attack:

There are generally two steps involved in developing any cryptographic primitive. First it is defined as an abstract mathematical object. Thereafter this mathematical

entity needs to be implemented in form of a program and in some cases these programs are further implemented in some specific hardware. These programs after implementation will be executed in a computing environment on processing units. These executions will present some specific characteristics. Side channel Analysis (SCA) refers to the attacks based on the physically observable characteristics during execution. Some of the common physical characteristics that are used for Side Channel Analysis are Power and Microprocessor time required for execution, electromagnetic radiation, heat dissipation and noise of the system etc. On the basis of above characteristics; there are different Side Channel attacks on ciphers in general and on stream cipher in particular and these are Simple Power analysis attack and Differential Power Analysis attack [FGK07,KJJ99], Timing Analysis attack [DK198,KOC96], Electromagnetic Analysis attacks [GMO01,QS01,AAR02] and Acoustic Cryptanalysis [ST13] are generally used and powerful techniques for Side Channel analysis attacks. Though there is no general countermeasure to these attacks but some of the possible countermeasures maybe noise addition, buffering of the output sequence, Physical shielding, reduction of signal size, eliminating the branch processing in implemented algorithm that will make the encryption time equivalent [ZF05, SX10].

2.3.1.3 Time Memory Tradeoff Attack:

A time memory tradeoff attack is a method of cryptanalysis that aims to attack a cryptographic primitive with lower complexity than look up table and an online complexity lower than exhaustive key search. TMTO is an improvement to the exhaustive key search attack that trades off computational time against memory complexity. This attack can be divided into two phases; an offline phase or pre-computation phase and online phase. In offline phase a table is constructed in like lookup table method by selecting different random keys and generating the output for each chosen key. These pairs of output strings and keys are stored in an indexed table indexed by the output strings. In the second phase or online phase, the attacker observes the output generated by unknown keys. Then these outputs are matched with the outputs of the table generated in the offline phase. If a match is found then corresponding key will be the key off the matched output.

Amirazizi and Hellmen were the first to propose Time memory processor trade-off attack [AH88] on block ciphers and in case of stream ciphers, TMTO was proposed by Babbage [BAB95] in 1995 and Golic [GOL97] in 1997 independently. Later on Biryukov and Samir combined Babbage and Golic scheme with Hellmen attack [BS00]. This attack was further refined by Birykov, Shamir and Wagner and applied on A5/1 [BSW78].

2.3.1.4 Distinguishing attack:

The idea of this attack was introduced by Fluhrer and McGrew on alleged RC4 key stream generator [FM00] and can be defined as any form of cryptanalysis where the attacker can extract some information from encrypted data sufficient to distinguish it from random data. The most important criterion for a good stream cipher design is that keystream generation should to be random. Distinguishing attack tries to identify the relations between internal state variables and output keystream. The internal structure of a cipher has to be analyzed extensively for distinguishing attack. Ciphers are required to use sufficiently long keystream to avoid distinguishing attacks.

2.3.1.5 Algebraic attack

Algebraic attacks are relatively new attacks for stream ciphers and progress is rapidly taking place in this field. Algebraic attacks are very much effective against LFSR based ciphers. The basic principle of algebraic attacks is to model a cryptographic system in terms of algebraic equations. The first step of this attack is to find the set of algebraic equations that links the initial state with the output keystream. Then keystream bits are observed and these values are substituted into the equations. Attackers try to collect maximum possible keystream bits. Then finally this system of equations is solved to determine the initial state and then derive the secret key from it. Algebraic Attack on stream cipher was first proposed by Courtois in [COU02] against Toycrypt. Later on Courtois further enhanced this attack and proposed Fast Algebraic attack [COU03] that was further strengthened by Armknecht [ARM04]. The idea behind fast algebraic attack was to get equations of lower degree by linearly combining the equations before solving the system of equation that drastically increases the speed of the attack.

2.3.1.6 Correlation attacks

Correlation attacks are widely applicable to stream ciphers, especially to design based on feedback shift registers. A correlation attack tries to extract some information about the initial state from the output keystream by exploiting the weaknesses in the combining function of the design.

Siegenthaler first introduced the Correlation attack against combination generator [SIE85] in 1985 but this attack was further improved by Meier and Staffelbach in 1988 as Fast Correlation Attack [MS88]. Zhang and Feng proposed an improved fast correlation attack on stream ciphers in [ZF09]. Correlation-immune functions need to be implemented for avoiding such attacks. In the case of LFSRs, the irregular clocking is one of the concepts to avoid linearity that will help in countering this attack.

2.3.1.7 Guess-and-determine attacks:

Guess and determine attacks are general attacks on stream ciphers. As it is clear from the name, in Guess and determine attacks, an attacker guesses a part of the internal state and tries to recover the full value of internal states by observing the keystream using the guessed part and small amount of known keystream. Finally a part of keystream is generated using the guessed values and then it is compared with the known keystream to check the correctness of the guessed values. In [MAT06] guess and determine attack was given against Polar Bear. Guess and determine attacks were also presented in [HR02] against SNOW. By irregular clocking, resistance against guess and determine attack can be increased. Guess and determine attacks are more effective against word oriented stream ciphers

2.3.1.8 Linear Masking attacks:

Linear masking attacks can be applied to those ciphers where there exist some non-linear process resembling block cipher design and in which linear masking is used to hide this process. In this attack first of all a non-linear characteristic is distinguished that is exhibiting some bias. Then linear process is analysed and some linear

combination is missing. The same linear combinations are applied to the cipher output and an attempt is made to find the traces of distinguishing property. Coppersmith et al. in [CHJ02] described a generic attack on stream ciphers using linear masking. Watanabe et al. proposed linear masking attack on SNOW [WAT04].

2.3.1.9 Related Key Attack:

In an attempt to provide a little bit of extra safety or security, some of the cryptographic protocol limits the amount of data which can be encrypted using a single key. In such cases either the new key is generated with using the IV (initialization vector) and with master key or to change the IV which in turns change the cipher key. In such type of ciphers, if the rekeying strategy relates the inputs to the internal states without sufficient non-linearity, then the cipher may become prone to a related key attack. These types of weaknesses are not very common in case of stream ciphers but there are some examples of related key attacks. Fluhrer et.al. in [FMS01] showed the related key attack on RC4 by exploiting a weakness of invariance in the key initialization algorithm. Sekar et al. presented a related key attack on Py-family of stream ciphers [SP071, BSF11].

2.3.1.10 Divide and conquer attack

Divide and conquer is a common technique based on dividing the problem into smaller problems and try to solve it in step by step procedure. Divide and conquer attack is based on same strategy where in a cipher is partitioned into smaller components and only a few key bits are determined in each stage. The attack can be termed as successful only if complexities of all the stages are smaller than the exhaustive key search. This concept was originally pointed out by Siegenthaler [SIE85]. High correlation immunity decreases the vulnerability to divide and conquer attack [SIE86].

This chapter gives an insight of cryptology and its related concepts and begins with an introduction of cryptography and its classification. Further the symmetric key cryptography and different modes in which symmetric key cryptography operates is discussed. Since this thesis is devoted to stream ciphers, this chapter presents some

basic concepts of stream cipher here and will be discussed in detail in next chapter. At the end, a study of existing cryptanalytic attacks on stream ciphers is presented.

Building stream cipher

Chapter 3

Building Stream Cipher

Stream cipher is a symmetric encryption algorithm, which typically operates on individual units of plaintext. A stream cipher generates a pseudo random sequence of digits known as the keystream. which is used to encrypt the plaintext in order to obtain the ciphertext. Encryption is accomplished by combining the keystream with the plaintext, usually with the bitwise XOR operation. The central issue for the security of stream ciphers is the method in which the keystream is generated. A stream cipher comprises of a state update function and an output function. The state of cipher is an allocation of memory which at time t uniquely represents the values of the set of variables that describes the current status of the cipher and is updated continuously during encryption, in order to encrypt bits at different positions in a message with different states. The output function is responsible for generation of keystream bits from the state and also performs the operation of encryptinn or decryption. An initial state of the stream cipher is normally determined by the secret key and a publicly known Initialization Value or Initial vector (IV).In an attempt to generate next state, the cipher outputs some bits known as the keystream. and then elevates to the next state where the process is repeated. The transition from one state

to the next is controlled by a clock. The keystream produced should be as random looking as possible in order to make it more resistant to attacks. Such a sequence is also called a pseudorandom sequence.

The main objective of a stream cipher is to produce a pseudorandom sequence of symbols known as the keystream. A pseudorandom sequence is a deterministically generated sequence that exhibits the characteristics of a truly random sequence. The characteristics that define a random sequence are determined through statistical measures and binary digit bias measurements. Desired properties of pseudorandom sequence are a long period and uniform distribution [ENG07]. The generated keystream is then mixed sequentially with the plaintext to produce ciphertext.
Stream ciphers are normally faster than block ciphers and do not cost more to implement in terms of hardware gates or memory. They have also limited error propagation, if the encrypted data is corrupted on the channel.

A stream cipher operates in two phases:

I. ***Setup phase:*** In the setup phase, the state of cipher is initialized by a scheduling procedure with the aid of the key and possibly an initialization vector. An initialization procedure is then performed to make sure that all IV bits and key bits are properly mixed and spread across the entire state.
II. ***Encryption/decryption phase:*** In this phase, stream cipher generates key stream symbols as a function of an internal state and secret key. This keystream is then combined with the message to form the cipher text. Decryption is done in a similar fashion.

3.1 Stream Cipher Classification

Stream ciphers, commonly on the basis of keystream generation, are classified as synchronous or self-synchronous stream cipher. In *synchronous stream ciphers*, the next state of the cryptosystem is yielded independent of both the plaintext and ciphertext, while as in *self-synchronized stream ciphers*; keystream generation depends on the collection of plaintext and ciphertext. As already mentioned in the

introduction of stream ciphers, stream ciphers suffer very limited error propagation. In case of synchronous stream ciphers, If an element of the ciphertext sequence is modified accidently or intentionally (but not deleted), then only the corresponding plaintext element is affected in the deciphered plaintext, provided that synchronization is maintained. In self-syochronous stream ciphers, an insertion, deletion, or change in ciphertext characters results in loss of only a fixed number of deciphered plaintext characters, after which the deciphering self-synchronizes. A ciphertext error in transmission affects at most t characters of the deciphered plaintext.

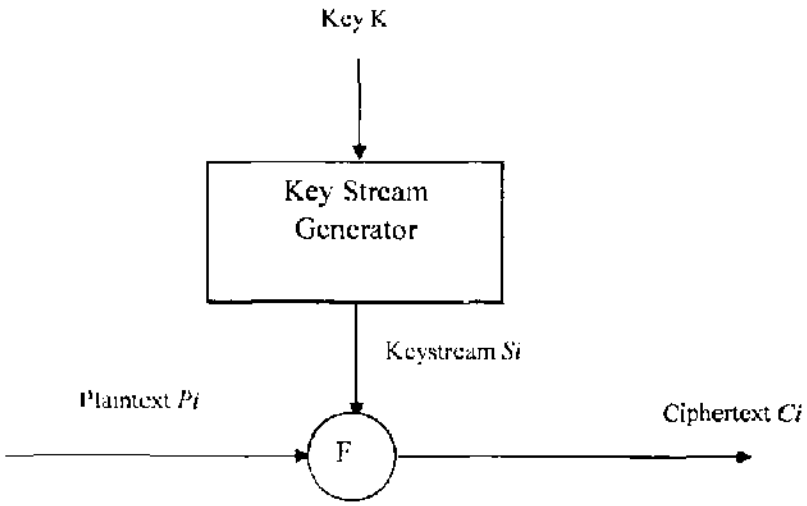


Fig 3.1: Synchronous Stream Cipher

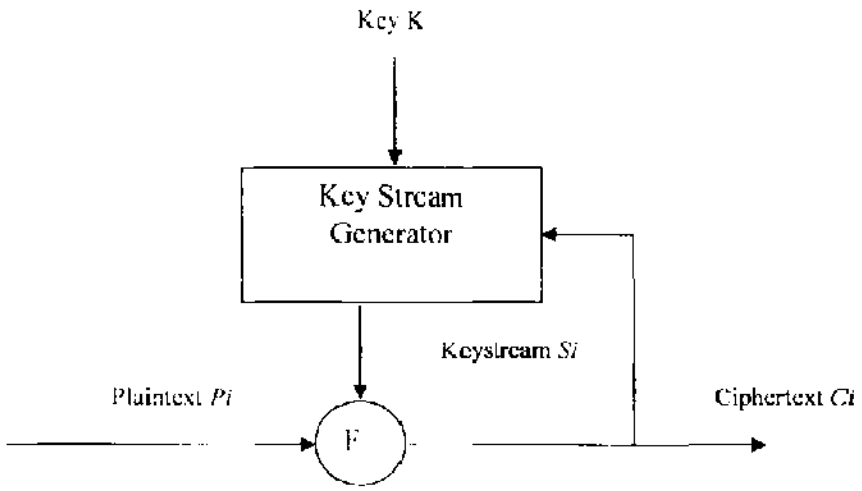


Fig 3.2: Self-Synchronous Stream Cipher

3.2 Random Number Generation

Cryptographic applications require uniformly distributed random strings, but physical random processes usually aren't perfect because they generate highly unpredictable, yet not uniformly distributed, values. Fortunately, the mathematical and computer science communities have developed ways to extract pure randomness from high-entropy sources

3.2.1 Randomness: A colloquial definition of random process is one whose consequences are unknown. Since randomness provides a way to create information that an adversary can't learn or predict, this attribute makes randomness crucial in cryptographic applications. It then becomes the task of good protocol designer to leverage this power in the best possible way to protect data and communication. The following two criteria [RAN13] are used to validate that a sequence of numbers is random:

3.2.1.1 Uniform distribution: The distribution of numbers in the sequence should be such that the frequency of occurrence of each of the numbers should be approximately the same. There are well-defined tests for determining whether a sequence of numbers matches a particular distribution, such as the uniform distribution

3.2.1.2 Statistically Independent: Each number drawn must be statistically independent of the others intuitively No one value in the sequence can be inferred from the others. There is no such test to "prove" independence. Rather, a number of tests can be applied to demonstrate if a sequence does not exhibit independence. The general strategy is to apply a number of such tests until the confidence that independence exists is sufficiently strong.

Generally there are two main approaches to random number generation using a computer: Pseudo-Random Number Generators (PRNGs) and True Random Number Generators (TRNGs) [RAN13]. Although there is one more noted approach termed as

Cryptographically Secure Pseudorandom Number Generators (CSPRNG) [PRE09]. The approaches have quite different characteristics and each has its pros and cons

3.2.2 Pseudo Random Number generator

Pseudo random number generators (PRNGs) use deterministic algorithm to generate numerical sequences by starting with a short random number string (a "seed") and then expand it into a much longer "random-looking" sequence. Though the output sequence is not statistically random, however, if the algorithm is good, the resulting sequences will pass many reasonable tests of randomness. Such numbers are referred to as pseudorandom numbers. It is important to note that a PRNG is a deterministic process [HVT05]: put back in the same state, it will reproduce the same sequence, as will two PRNGs initialized with the same seed. This property makes PRNGs suitable for use as stream ciphers. It can be shown that a PRNG is necessarily periodic: the sequence it produces will repeat itself after a (possibly extremely long) period.

Pseudo-random number generators must satisfy all the following conditions:

(a) The statistical properties are close to that of a true random number. A past or future unknown output bit is hard to predict from the known output bit history.

(b) The seed size must be large enough to be secure against an exhaustive key search of the system that uses a pseudorandom number generator.

(c) The statistical properties of pseudorandom number generators must pass a typical statistical test suite for randomness. [HVT05]

3.2.3 True Random Number Generators (TRNG)

True random number generators (TRNGs) are characterized by the fact that their output cannot be reproduced and sequences of numbers produced are not predictable. The characteristics of TRNGs are quite different from PRNGs. First, TRNGs are generally rather *inefficient* compared to PRNGs, taking considerably longer time to produce numbers. Also unlike Pseudorandom Number Generators (PRNGs), True random number generators are *nondeterministic*, meaning that a given sequence of numbers cannot be reproduced, although the same sequence may of course occur several times by chance. TRNGs have no period [RAN13]

3.2.4 Cryptographically Secure Pseudorandom Number Generators (CSPRNG)

Cryptographically secure pseudorandom number generators (CSPRNGs) are a special type of PRNG which possess the additional property of unpredictability. Intuitively, A PRBG is a cryptographically secure random number bit generator (CSPRBG) if for the generated output sequence; it can be proven that no polynomial-time algorithm exists to solve the next bit test. Informally, this means that given n output bits of the key stream $s_i, s_{i+1},...,s_{i+n-1}$, where n is some integer, it is computationally infeasible to compute the subsequent bits $s_{i+n}, s_{i+n+1},....$ A more exact definition is that given n consecutive bits of the key stream, there is no polynomial time algorithm that can predict the next bit s_{n+1} with better than 50% chance of success. Another property of CSPRNG is that given the above sequence, it should be computationally infeasible to compute any preceding bits s_{i-1}, s_{i-2}. It is important to note here that the need for unpredictability of CSPRNGs is unique to cryptography. In virtually all other situations where pseudorandom numbers are needed in computer science or engineering, unpredictability is not needed.

3.3 Basic Building Blocks of stream cipher

This section describes some commonly used building blocks for the construction of stream ciphers. The chapter starts with a description of a linear feedback shift register, which is a very popular building block in stream cipher primitives and then the focus is shifted to Non-linear feedback shift registers, which gives us an opportunity to avoid the inherent linearity faced in LFSRs. Following this, a discussion is presented on various methods of introducing nonlinearity.

3.3.1 Linear feedback shift register

Linear Feedback Shift Registers (LFSRs) have always received considerable attention in cryptography. Owing to the good statistical properties, large period and low implementation costs, LFSR have achieved wide acceptance in developing stream ciphers. An LFSR is a shift register that, using feedback, elevates the bits through the

register from current location to the next most-significant location, on each rising edge of the clock. Selected outputs (taps) are combined in an exclusive-OR (or exclusive-NOR) fashion to form a feedback mechanism, which causes the value in the shift register iterate endlessly through a sequence of unique values. An LFSR of any given size n (number of registers) is capable of producing every possible state during the period $N=2^n-1$ excluding the all-zero state, such a sequence is called maximal sequence (abbreviated as m-sequence).Linear feedback shift registers as maximal length sequence generators are widely used in stream ciphers for key stream generation due to their good statistical properties, large period, low implementation costs and are readily analysed using algebraic techniques. Maximal length sequences are generated when the LFSR passes through every non-zero state once and only once and are obtained when the feedback polynomial to which LFSR corresponds is primitive [JJD09], a feedback polynomial of degree n is primitive if it is irreducible (cannot be factored) and has a period equivalent to 2^n-1.

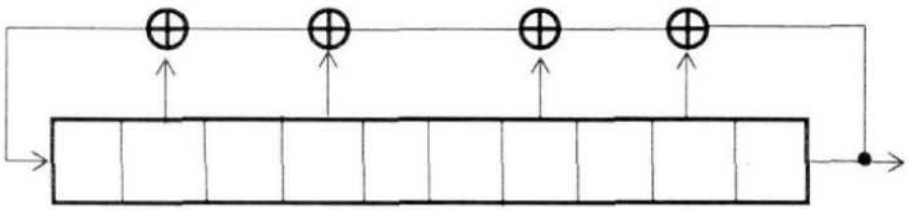


Fig3.3: Linear feedback shift Register (LFSR)

The predominant characteristic like large linear complexities, large period, statistical properties and pseudo randomness of the key stream generated by LFSR make LFSR a good choice for developing stream ciphers besides allowing a ready algebraic analysis of keystream generated by linear feedback shift registers

3.3.1.1 Mechanism of an LFSR:

The implementation of Linear feedback shift register consists of n input shift registers, where the input bit is calculated as a linear function of the content of the register. An n stage LFSR consists of clocked storage elements in the form of a shift register S and a feedback path in the form of tap sequence T where shift register $S=(s_n,s_{n-1},s_{n-2},\ldots\ldots,s_1)$ and a tap sequence $T=(t_n,t_{n-1},t_{n-2},\ldots\ldots,t_1)$, with each s_i and t_i

being one binary digit. At each clock interval, all the bits are shifted right except bit s_1, which is appended to the key stream, and a new bit derived from S and T is fed back as input to the left end of the register. Whether a feedback is active or not is determined by feedback coefficients $T=(t_n, t_{n-1}...t_1)$

- If $t_i = 1$, feedback is active
- if $t_i = 0$, feedback is passive

A Linear feedback function produces a sequence S, satisfying the linear recurrence function. Assuming that the LFSR is initially loaded with some seed value s_0, s_1, $s_2...s_{n-1}$. The next output bit is computed by the XOR-sum of products operation of storage elements and corresponding taps:

$$S_n \equiv s_{n-1}t_{n-1} + \cdots + s_1t_1 + s_0t_0 \ mod\ 2$$

Similarly next output is:

$$S_{n+1} \equiv s_nt_{n-1} + \cdots + s_2t_1 + s_1t_0 \ mod\ 2$$

Finally the general output can be shown as:

$$S_{n+1} \equiv \sum_{j=0}^{n-1} t_j \cdot S_i + j \ mod 2$$

Since the number of recurring states is finite, the generated sequence produced by LFSR must repeat itself after a finite period and also the length of the sequence is completely determined by the feedback coefficients and seed value.

Since an n-bit vector can assume only 2^n-1 states excluding an all-zero state, An n-bit LFSR can deterministically assume its next state based on its previous state, as a result of which, as soon as an LFSR encounters a previous state, It starts to repeat itself. Therefore the maximum sequence length without repetition is 2^n-1. An all-zero state is discarded because if an LFSR assumes this state, it will get "stuck" and will never be able to leave this state.

An LFSR with a feedback coefficient vector $T(t_{m-1}...t_1, t_0)$ is often specified with the help of polynomial:

$$P(x) = x^m + p_{m-1}x^{m-1} + \cdots p_1x + p_0$$

In order to have a maximal length sequence, the feedback polynomial of the LFSR should be primitive. Primitive polynomial is an irreducible polynomial of degree n, whose period is 2^n-1. The degree of the polynomial is the length of the shift register. An important aspect of irreducible and primitive polynomial is that all the primitive polynomials are irreducible but the reverse is not true. For example,

the polynomial $x^4 + x+1$ is irreducible as well as primitive whereas the polynomial $x^4+x^3+x^2+x+1$ is irreducible but not primitive[WIN08]. An irreducible polynomial p(x) of degree n is not said to be primitive if its order is not $2^n - 1$. There may be more than one primitive polynomial of order n>2. The number of such primitive polynomials can be determined through the factorization theorem. The number of primitive polynomials of degree n is:

$$\frac{\emptyset(2^n - 1)}{n}$$

Where$\emptyset(x)$, known as the Euler function, denotes the number of positive integers less than the integer x and relatively prime to it. [ZYW91]

Modification of the feedback scheme allows us to implement LFSR in two different variations. Though the variations are cryptographically no better, but it can still affect the periodicity and software implementation [SCH96]. Based on the configuration of gates and registers, LFSR can be divided in two categories.

- *The Fibonacci LFSR, also known as External-XOR LFSR or just LFSR*
- *The Galois LFSR, also known as Internal-XOR or canonical LFSR*

In the Fibonacci implementation, the taps are XORed sequentially with the output bit and the fed back into the leftmost bit. The shift register is initially loaded with bits a_0, $a_1 \ldots a_{r-1}$ called the seed value (any value except all zeroes) and then clocked while as in the Galois implementation, with each clock cycle, bits that are not taps are shifted one position to the right unchanged. The taps on the other hand, are XORed with the output bit before they are stored in the next bit. The shift register is initially loaded with bits a_0, $a_1 \ldots a_{r-1}$ called the seed value (any value except all zeroes) and then clocked.

3.3.1.2 Desirable properties of LFSR-based keystream

While designing cryptosystem it becomes imperative for cryptographers to consider suitable criteria for keystream generator to be used in cipher. Some of these design criteria are large linear complexity, large period and good statistical properties

a) Linear complexity:

It is defined as length n of shortest LFSR that can mimic the generated output [AE97]. It is an indication for how difficult a sequence might be to replicate. While a high linear complexity is a necessary condition, it is not a sufficient condition.

b) Period:

For an L stage LFSR, period is defined as the length of the stream, before it repeats itself. An LFSR with short Period results in encryption of different parts of plaintext with same keystream, which causes severe weakness. Practically, the period should be long enough to accommodate entire plaintext without repeating the keystream. The longest period possible corresponds to the largest possible state space, which is produced by a maximal length tap sequence. Maximality of period guarantees good statistics. [ZYW91]

c) Statistical measures:

Once the sequence has been generated, it is required to assess it statistically, how well it is generated. Several statistical tests exist to determine the statistical behavior of the sequence. These include run test, rank test, frequency test, Fourier transform test, serial test, Lempel-Ziv complexity test and linear complexity test. [CB86] These tests generally check for random distribution, linear dependence among fixed length substrings, distribution of ones and zeroes in a sequence, the level of compression that can be carried out on tested sequence and whether a sequence is complex enough to be considered random

LFSRs are notoriously insecure from a cryptographic standpoint because the structure of an n-bit LFSR can be easily deduced by observing $2n$ consecutive bits of its sequence using the Berlekamp-Massey algorithm [DUB09]. Although Properties like large period, large linear complexity and a good statistical behavior are necessary but are not sufficient condition for a stream cipher to be considered cryptographically secure. Due to the inherent linearity, LFSR based stream ciphers are susceptible to several general attacks including known plaintext attack [PP10], algebraic attack [VOR07], cache timing attack [LZH09], fast correlation attack [CAN05].

3.3.2 Nonlinear feedback shift register

Non-Linear Feedback Shift Registers (NLFSRs) have been proposed as an alternative to Linear Feedback Shift Registers (LFSRs) for generating pseudo-random sequences for stream ciphers. A Non-Linear Feedback Shift Register (NLFSR) consists of n binary storage elements, called bits. Each bit i has an associated state variable x_i which represents the current value of the bit i and a feedback function f_i which determines how the value of i is updated. A state of an NLFSR is an ordered set of values of its state variables $(x_0, x_1, \ldots x_{n-1})$. At every clock cycle, the next state is determined from the current state by updating the values of all bits simultaneously to the values of the corresponding f_i. NLFSRs have been shown to be more resistant to cryptanalytic attacks than LFSRs. However, construction of large NLFSRs with guaranteed long periods remains an open problem.

Cryptographically strong pseudo-random sequences are produced by combining more than one LFSR with some method to introduce non linearity. Three general methods of combining LFSRs employed to overcome the problem of linearity in LFSR based stream ciphers are:

- Nonlinear combination generator
- Nonlinear filter generator
- Clock-controlled generator

3.3.3 Non Linear Combination Generator

The key stream is generated by manipulating the outputs of several parallel LFSRs using a nonlinear Boolean function f. The function f is called the combining function and maps one or more binary input variables to a binary output variable. The Boolean function must have a high algebraic degree, high nonlinearity and preferably a high order of correlation immunity. The keystream generated z is given by z = f(x1,x2 .... xn), where x1,x2....xn are the outputs of n-sub generators. General model for a nonlinear combination generator is shown in figure below

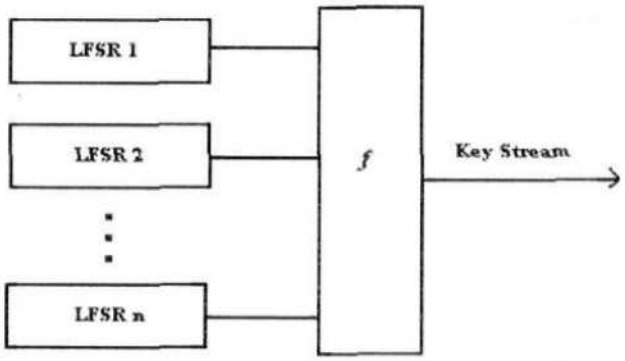


Fig 3.4: Nonlinear combination generator

If n maximum-length LFSRs with lengths $l_1, l_2, \ldots l_n$ are used together with the Boolean function f, the linear complexity of the keystream is

$$f(l_1, l_2, \ldots l_n) = a_0 + a_1 l_1 + \ldots a_n l_n + \ldots a_{12\ldots n} l_1 l_2 \ldots . l_n$$

where $a_0, a_1 \ldots a_n$ are the coefficients of in the algebraic normal form of f.[WIN08]

3.3.4 Non Linear Filter Generator

A nonlinear filter generator uses a single maximum-length LFSR, and the keystream is generated as a nonlinear function f of the state of the LFSR.

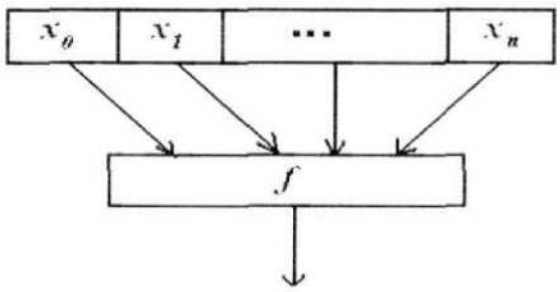


Fig 3.4: Nonlinear filter generator

The function f is called the filtering function. In this generator a single maximum length LFSR is used in contrast to the nonlinear combination generator where several LFSRs were used and different stages of single LFSR are used as input t the filtering function f. If a nonlinear filter generator is constructed by using a maximum-length LFSR of length L and a filtering function f of nonlinear order m then the linear complexity of the keystream is at most:

$$L_m = \sum_{i=1}^{m} \binom{n}{i} \qquad \text{[OZG06].}$$

In particular, here are some problems with nonlinear-feedback shift register sequences.

- There may be biases, such as more ones than zeros or fewer runs than expected, in the output sequence.
- The maximum period of the sequence may be much lower than expected.
- The period of the sequence might be different for different starting values
- The sequence may appear random for a while, but then "dead end" into a single value.

(This can easily be solved by XORing the nonlinear function with the rightmost bit.)
On the plus side, if there is no theory to analyze nonlinear-feedback shift registers for security, there are few tools for cryptanalysis of stream ciphers based on them. Nonlinear-feedback shift registers can be used in stream-cipher design, but it should be used carefully.

3.3.5 Clock Controlled Generator

In clock controlled generators, the movement of data of one LFSR is controlled by the output of another LFSR. The register enabling clocking control is called as control register, and denoted as CR. The register which generates keystream according to the output sequence of CR is called as generator register, and denoted as GR. If *a(i)* represent the bit produced by CR and *b(i)* represent the bit produced by GR at instant *i*, the output of the keystream generator at time *i* is given as

$$u(i) = \sum_{k=1}^{i} a(k) \qquad \text{[JJD09]}$$

Since GR is clocked in an irregular manner, the output is the nonlinearly decimated sequence of a regularly clocked generator. General model for a nonlinear combination generator is shown in figure 3.5:

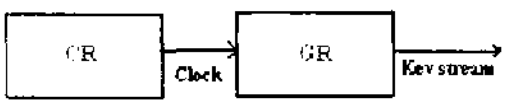


Fig 3.5: Clock controlled generator

3.3.6 Boolean function

Cryptographically secure pseudorandom number generators are of vital importance for stream cipher design and LFSRs though not secure due to their inherent linearity, are commonly used as part of key stream generators in stream ciphers due to their good statistical properties, large periods and low implementation costs.

Boolean functions are the building blocks of symmetric cryptographic systems. Boolean functions used in cryptographic applications have to satisfy various cryptographic criteria. Although the choice of the criteria depends on the cryptosystem in which they are used, there are some properties (balancedness, nonlinearity, high algebraic degree, correlation immunity, propagation criteria) which a cryptographically strong Boolean function ought to have. A Boolean function is said to be balanced if its truth table has equal number of ones and zeros. A Boolean function is said to be correlation immune of order m, if the output of the function is statistically independent of the combination of any m of its inputs. The non-linearity of a Boolean function can be defined as the distance between the function and the set of all affine functions. In stream ciphers, they are usually used to combine the outputs to several linear feedback shift registers, or to filter (and combine) the contents of a single one. The sequence of their output, during a certain number of clock cycles, then produces the pseudo-random sequence that is used in stream ciphers.

3.3.7 S-BOX

An S-Box (Substitution-box) is a basic component of symmetric key algorithms used for substitution, that generally takes m bits as input and transforms them into n number of output bits , where n is not necessarily equal to m.

$$\text{S-Box } S_i\text{: } \{0, 1\}^m \rightarrow \{0, 1\}^n$$

An m x n S-Box can be implemented as a lookup table with 2m words of n bits each. Fixed tables are normally used, as in the DES, but in some ciphers the tables are generated dynamically from the key which performs substitution These S-boxes are carefully chosen to resist linear and differential cryptanalysis [HEY02]. One obvious characteristic of the S-box is its size. An m x n S-box has m input bits and n output

bits. Larger S-boxes, by and large, are more resistant to differential and linear cryptanalysis [SCH96]. On the other hand, larger the dimension n (exponentially), larger the lookup table. Thus, for practical reasons, a limit of n equal to about 8 to 10 is usually imposed. Another practical consideration is that the larger the S-box, the more difficult it is to design it properly

Design Criteria of Good S-Box

- Balanced Component functions
- Non-linearity of Component functions high
- Non-zero linear combinations of Component functions balanced and highly non-linear
- High Algebraic degree

Nyberg, who has written a lot about the theory and practice of S-box design, suggests the following approaches [ROB95]:

- **Random:** Use some pseudorandom number generation or some table of random digits to generate the entries in the S-boxes. This may lead to boxes with undesirable characteristics for small sizes (e.g., 6 x 4) but should be acceptable for large S-boxes (e.g., 8 x 32).
- **Random with testing:** Choose S-box entries randomly, then test the results against various criteria, and throw away those that do not pass
- **Human-made:** This is a more or less manual approach with only simple mathematics to support it. It is apparently the technique used in the DES design. This approach is difficult to carry through for large S-boxes
- **Math-made:** Generate S-boxes according to mathematical principles. By using mathematical construction, S-boxes can be constructed that offer proven security against linear and differential cryptanalysis, together with good diffusion.

This chapter has elaborated upon stream ciphers and had a closer look on various techniques of stream cipher design. Due to their distinct properties, stream ciphers are suitable for several applications. Most notably, they are usually faster and have a lower hardware complexity than block ciphers. They are also appropriate when buffering is limited, since the digits are individually encrypted

and decrypted. Moreover, synchronous stream ciphers are not affected by error-propagation.

Analysis of existing stream ciphers

Chapter 4

Analysis of Existing Stream Ciphers

Stream ciphers are cryptographic primitives that ensure the confidentiality of communications. This chapter is intended to provide an insight of several stream cipher proposals found in literature. The existing stream ciphers can roughly be classified as ciphers either optimized for software implementation or optimized for hardware implementation. In the former case, the ciphers typically require few CPU instructions to compute one key stream bit. In the latter case, they tend to be based on operations which can easily be realized in hardware. A third class of stream ciphers is realized by using block ciphers as building blocks. The cipher feedback mode, output feedback mode and counter mode that were introduced in Chapter 2 are examples of stream ciphers derived from block ciphers. Even though many stream ciphers have been proposed over the years, there are considerably fewer well-investigated ones. The security of many proposed stream ciphers is unknown, and many stream ciphers have been broken. Different stream ciphers dealt in this chapter have been part of high profile stream cipher projects like eSTREAM [ECR10] and NESSIE [NES12].

4.1 SOBER family of stream ciphers

SOBER, a contrived acronym for seventeen octet byte enabled register, is a family of stream ciphers, widely used in embedded devices and was originally proposed by G. Rose in 1998 . The SOBER family ciphers include SOBER [ROS198], SOBER-II [ROS298], S16, S32 [ROS00], SOBER-t8, SOBER-t16, SOBER-t32 [ROS398] and SOBER-128[HR032]. SOBER [ROS198] first member of the family, is a software stream cipher that was designed with the intent of meeting the essentials of embedded applications such as voice encryption in wireless telephones which place severe constraints on the amount of processing power, program space and memory available for software encryption algorithms. Several variants of SOBER have been developed to strengthen its security or to be suitable for 16-bit and 32-bit processors. When various weaknesses were found in the original design of SOBER[6], it was superseded by SOBER-II and two SOBER variants: S16 and S32 [ROS498], The analyses of SOBER-II found attacks that are specific to the overall structure of the cipher, rather than exploiting a weakness of the individual components used in the cipher.

S16 copies the structure of SOBER-II[ROS498] and is an enhancement of it to 16-bit wide arithmetic, while S32 is an extension of the design principles of SOBER to 32-bit wide arithmetic[ROS398, HR00] and its structure is different from SOBER-II's and S16's. Nevertheless, there were opportunities for strengthening SOBER-II and S16 that could not be ignored. Consequently, three new versions based on 8-bit, 16-bit and 32-bit operations, called the t-class of SOBER ciphers [HR00] were developed as a substitute to SOBER-II, S16 and 32-bit version. The synchronous stream ciphers SOBER-t16 and SOBER-t32 were submitted to the NESSIE program [BL11]; as a stream cipher for 128-bit key and 256-bit key strength respectively. though both ciphers were found to fall short of the stringent NESSIE requirements but according to final NESSIE security report released in Feb 2003, there were only four stream cipher primitives which were considered during the phase II: BMGL[ZJZ09],SNOW[EJ00], SOBER-t-16 and SOBER-t-32[HR00].Subsequent to NESSIE, SOBER-128 [HR032] is an enhanced version of SOBER-t32. The modifications directly address the concerns arising in the analyses of the t-class ciphers. The 128-bit key strength proposed for SOBER-128 is reduced from the 256-

bit key strength proposed for SOBER-t32 to ensure that SOBER-128 provides far in excess of the stated security level. SOBER-128 was developed as a very conservative stream cipher, but with innovative (and flawed) message integrity functionality [HR032].

In the recent past enhancements to sober 128 have been developed in the form of NLS, NLSv2, SHANNON, TURING, and BOOLE.

4.1.1 Description of SOBER

The SOBER families of ciphers are based on the same principle and have almnst similar structure. Nearly all SOBER family ciphers consists of three basic components [HR032]. There is a Linear Feed-back Shift Register (LFSR), which uses a recursion formula to generate the stream of pseudo-random bits. Next a Non-Linear Function (NLF) combines these words in a non-linear way to produce the NLF-stream v_n. Finally, the so-called stuttering produces the keystream z_i by decimating the NLF-stream in an irregular fashion.

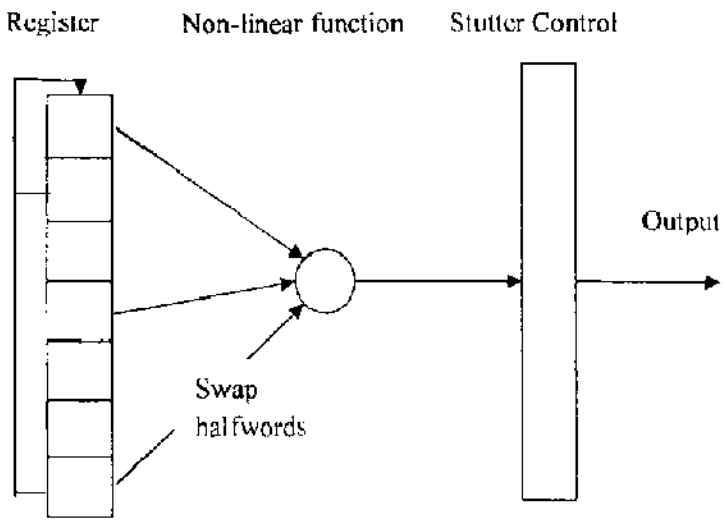


Figure 4.1: Block structure of SOBER family ciphers

4.1.1.1 Linear Feedback Shift Register (LFSR)

The SOBER family cipher's LFSR's are typically based on recurrence relation over the Galois Field of order 2 GF (2w), W $\in$ {8, 16, and 32}, the output sequence (of bits) is defined by

$$S_{n+k} = C_{k-1}S_{n+k-1} + C_{k-2}S_{n+k-2} + \ldots + C_1S_{n+1} + C_0S_n \text{ [ROS398]}$$

Where Sn is the n-th output of the sequence, the constant coefficients c_i , $0 \leq i \leq k-1$,are elements of GF(2) , that is, single bits, and k is called the order of the recurrence relation.

The LFSR is typically represented by the polynomial

$$P_1(x) = x^k + c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1x + c_0 \text{ [ROS398]}$$

Where p1(x) indicates that multiplication and addition are over GF(21).

The length of the LFSR is 17, with every register containing one word, resulting in an internal memory of 544 bits. The contents of LFSR at time t is called the state of the LFSR and is denoted by a vector S

$$S_t = (s_t, s_{t+1}, s_{t+2}, \ldots s_{t+16}) = (r_0, r_1, r_2, \ldots r_{16})$$

The next state of the LFSR is calculated by shifting the previous state one step, and calculating a new word s_{t+17} as a linear combination of the words in the LFSR. The word $s_{t+17}$is calculated as follows:

$$s_{t+17} = s_{t+15} \oplus s_{t+4} \oplus \alpha \cdot s_t ,$$

With α being a constant

4.1.1.2 Non-Linear Function (NLF)

The non-linear function plays a vital role in strengthening the security of SOBER family by deterring the various attacks, that linear feedback and stuttering are susceptible to. The purpose of the NLF is to disguise the linearity in the LFSR stream. After every cycle of the LFSR, the NLF combines words from the register in a non-linear function; the outputs form the NLF stream v_t

The Non-Linear Function (NLF) at any instance of time t, combines five values denoted $s_n, s_{n-1}, s_{n-6}, s_{n-13}, s_{n-16}$ from certain positions of LFSR state, using a technique involving the rotation of partial sums and calculates one output, called v_t. This output can be written as:

$v_t = ((f(s_t \boxplus s_{t+16}) \boxplus s_{t+1} \boxplus s_{t+6}) \oplus K) \boxplus s_{t+13}$. [HR00]

In above equation, K is a key dependent constant used in t-class and succeeding ciphers of family and is derived immediately after the secret session key is loaded, $\boxplus$ denotes addition modulo2^{32}, $\oplus$ represents bitwise XOR and f is a non-linear function.

The function f serves three purposes [EJ02]. Firstly, it removes the linearity in the least significant bit and adds significant non-linearity to the remaining bits. Secondly, it ensures that the addition of r_0 and r_{16} does not commute with the addition of r_1 and r_6. Thirdly it ensures that every bit of the output of the NLF depends on every bit of r_0 and r_{16}. XORing the value of Konst into the NLF has two purposes. Firstly, it increasing the complexity of any attack (excluding exhaustive key search) as there are now 216 possible NLF functions. Secondly, the Konst is XORed (rather than added) so as to lower the probability of the addition of r_{13} commuting with the addition of r_1 and r_6. There is still a small probability that the operations will commute, but this probability is low and relies on the value of Konst.

The interior design of the function f used in case of SOBER-t16 and SOBER-t32 of is pictured in Figure 4.2[ROS398]. First the input is partitioned into a high part containing the 8 most significant bits, and a low part containing the remaining bits. The high part addresses an SBOX with W bits of output. The 8 most significant bits are directly taken as the f-function output, whereas the least significant part of the SBOX output is first XORed to the low part from the input.

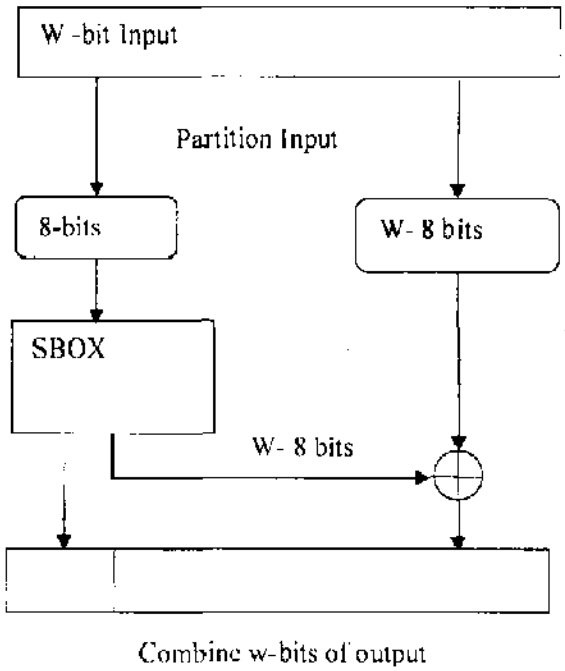


Figure 4.2: The structure of the *f*-function in SOBER-t16 and SOBER-t32

4.1.1.3 Stuttering

Stuttering has been considered as the most ad-hoc part of the design of the SOBER family ciphers [ROS398], but still appears in almost every variant of SOBER except SOBER 128 and offers the best increase in security. It introduces irregular decimation in the output produced by the NLF to avoid the reconstruction of the state particularly by correlation attack. In these implementations, occasional words of nonlinear output are used to determine the inclusion of other words in the output stream. When the generator is started, the first output word is taken to be used as a stutter control word (SCW)[BCL03]. Each stutter control byte is broken into pairs of bits called dibits, with the least significant pair of bits being used first. The dibits provide the cipher with instructions regarding how many times to cycle the LFSR, whether to output an NLF output, and how this value is included in the key stream. When all dibits in the SCW have been used, the LFSR is clocked once and a new SCW is read from the output of the NLF.

There are four possible values for each of the two-bit stutter controls[ROS00] These determine the output from the stream generator as follows:

(0, 0) The register is cycled but no output is produced.

(0, 1) The register is cycled, and the nonlinear output XOR the constant C becomes the output of the generator. The register is then cycled again.

(1, 0) The register is cycled twice, and the (second) nonlinear output becomes the output of the generator.

(1, 1) The register is cycled, and the nonlinear output XORed with the complement of constant C becomes the output of the generator.

4.1.2 Comparative Analysis of Sober Family Members

This section carries out an analyses based on the design features like keying the stream cipher, byte order considerations, memory requirements, key strength, polynomials used, LFSR over GF (2^w), and security issues

4.1.2.1 Keying the Stream Cipher

a) Byte Order Considerations

SOBER,SOBER II,S16,S32,SOBER t-8,t-16 and t-32 utilize native processor operations on integer data items, but are expected to accept keys which are simply strings of bytes, and to produce a stream of bytes as output for encryption purposes.[ROS00]. To ensure compatibility between implementations running on different processors, a translation between native byte ordering and external byte ordering is necessary. Since all internet standards are defined using "big-endian" byte ordering[HEL85], in which the most significant byte of a multi-byte quantity appears first in memory, this is what is chosen for SOBER,SOBERII,S16,S32,SOBER t-8,16.

On "little-endian" machines, the key and the words of the output stream must be byte reversed before being loaded or XORed into the buffer, respectively

In contrast SOBER-128 is entirely based on 32-bit word operations internally [HR032], but the external interface is specified in terms of arrays of bytes. Conversion between 4-byte chunks and 32-bit words is done in "little-endian" fashion irrespective of the byte ordering of the underlying machine. This is a break with the tradition of previous members of the SOBER family

b) Key Loading

The SOBER family ciphers are designed for applications in wireless telephony. Loss of synchronization haunts such applications a great deal. In order to avert such calamitous loss, one solution adopted by GSM system is to have each encrypted frame implicitly numbered with a frame key, and the stream cipher re-keyed for each frame with the secret session key and the frame key. SOBER supports such a two tier keying structure [ROS198]. S16 shares this two-level keying structure. S32 was not designed to support such a two tier keying structure[ROS00]; All t-class ciphers and SOBER 128 have been designed to support the two-tiered keying structure in addition to the standard mode of operation[HR032]. The cipher is keyed and re-keyed by using operations that transform the values in the register under the influence of key material. Two principle operations are employed:

Include(X): adds the word X to r_{15} modulo 2^w;w={8,16,32}.

Diffuse(): cycles the register and this operation clocks the register, obtains the value by XORing the output of the NLF with r_4. Table 4.1 gives a comparison of key length, bit operation and memory requirements in bytes of different variants of SOBER family.

Table 4.1: The Key length, bit operations and memory requirements of SOBER family ciphers

Cipher	w-bit Operation	Max Key Length (bits)	Memory (bytes)
SOBER	8	128	54
S-16	16	194	108

S-32	32	256	78
SOBER t8	8	64	54
SOBER t16	16	128	105
SOBER t32	32	256	210
SOBER 128	32	128	140

4.1.2.2 Linear Feedback Shift Register over GF (2^w)

The LFSRs of SOBER, SOBER-II, SOBER-t8 are over GF (2^8); The LFSRs of S16, SOBER-t16 are over GF (2^{16})[ROS00]. The LFSRs of S32, SOBER-t32, and SOBER-128 are over GF (2^{32}). The length of S32 is 9 stages and the lengths of the others are all 17 stages. Table 4.2 compares the irreducible polynomials used by SOBER family ciphers and their hexadecimal representation

Table 4.2: The irreducible polynomial used in SOBER family ciphers

Cipher	Irreducible Polynomial	Hexadecimal Representation
SOBER	$X^8+X^6+X^3+X^2+1$	0x14D
S-16	$X^{16}+X^{14}+X^{12}+X^7+X^6+X^4+X^2+X+1$	0x150D7
S-32	$X^{32}+(X^{24}+X^{16}+X^8+1)(X^6+X^5+X^2+1)$	0x165656565
SOBER t8	$X^8+X^6+X^3+X^2+1$	0x14D
SOBER t16	$X^{16}+X^{14}+X^{12}+X^7+X^6+X^4+X^2+X+1$	0x150D7
SOBER t32	$X^{32}+(X^{24}+X^{16}+X^8+1)(X^6+X^5+X^2+1)$	0x165656565
SOBER 128	$x^{17}+X^{15}+X^4+\alpha$	0x170D

LFSR of various variants of SOBER are updated according to the recurrence relation shown in table 4.3:

Table 4.3: The feedback function of SOBER family ciphers

Cipher	Feedback Function
SOBER	$S_{n+17}=206 \boxplus s_{n+15} \oplus s_{n+4} \oplus 99 \boxplus s_n$
S-16	$S_{n+17}=E38216 \boxplus s_{n+15} \oplus s_{n+4} \oplus 673C16 \boxplus s_n$
S-32	$S_{n+9}=6327DFDA16 \boxplus s_{n+4} \oplus 2 \boxplus s_n$
SOBER t8	$S_{n+17}=CE16 \boxplus s_{n+15} \oplus s_{n+4} \oplus 6316 \boxplus s_n$
SOBER t16	$S_{n+17}=E38216 \boxplus s_{n+15} \oplus s_{n+4} \oplus 673C16 \boxplus s_n$
SOBER t32	$S_{n+17}= s_{n+15} \oplus s_{n+4} \oplus C2DB2AA316 \boxplus s_n$
SOBER 128	$Sn+17= s_{n+15} \oplus s_{n+4} \oplus \alpha \boxplus s_n$

4.2 Py family of Stream Ciphers

The stream cipher Py (Roo) analyzed in this chapter was one of these candidates in profile 1 stream cipher category. After initial analysis of less than a year 28 ciphers were promoted in phase 2 including Py. But some attacks were reported against it and to improve these shortcomings a new version Pypy was introduced but that was also susceptible to some attacks. The eSTREAM committee eliminated Py from phase 3 but the report [EST12] says that "Py and its variants demonstrate a promising approach that might offer exceptional performance. Unfortunately, however, there is sufficient analysis [IOKM07, PPS05, and WP06] to suggest that the submitted versions of the cipher demonstrate a weakness in the design."

To overcome the deficiencies in the previous variants of the Py, the designers had tweaked these ciphers and proposed three new ciphers TPy, TPypy, TPy6. Even though these tweaked variants have solved the problems of the previous known attacks against this family but new attacks were also reported against these new variants. In a new proposal Paul, Preneel & Sekar have introduced two new variants RCR-32 and RCR-64 and claimed it to be secure against any attack known for Py and its previous variants. But most of the claimed attacks against tweaked versions i.e.

TPy, TPypy& TPy6 were claimed as non-attack by the designers of these ciphers. This chapter has attempted to analyze the design and security specifications of Py cipher and its variants to assess the reliability of these ciphers and get a better understanding for the design of a stream cipher that is more secure as well as efficient. This chapter studies the design specifications, implementation details and security attacks of Py cipher and its variants.

4.2.1 Design Specifications of Py family

The main component in the design of Py (Roo) family of ciphers is rolling arrays. A rolling array is vector whose units are cyclically rotated and every rotation shifts its entries by one location. Some additional basic operations are performed as part of the array rotation. A very important property of the rolling array is that if the same entry is accessed in two consecutive steps then a different content will be generated. This property is very useful for increasing the speed of mixing the internal state of the cipher. In the Py (Roo) stream cipher and its other variants two rolling arrays have been used and both affect each other by their operation. Permutation and swap operations are performed on one array and then the update operation accesses the other entries. One rolling array is a permutation P of all 256 byte values and other is an array Y of 260 words of 32 bits [BS05]. All the entries are rotated and oldest entries is updated. In this way indirect access is performed that add a large amount of complexity to the mixing process that make it complex for a cryptanalyst to follow.

4.2.1.1 Py (Roo)

Py (pronounced Roo) is a synchronous stream cipher designed in response to eSTREAM project call. The main component of this cipher design is rolling arrays. It also uses various other ideas from many other ciphers, like permutation and variable rotation. To some extent Py is similar to RC4 as it also uses the technique of random scuffle [PPS05]. In Py all the array elements are also rotated in every round. The main strength of this cipher Py is its speed. It is 2.5 times faster than RC4. It takes less than 2.9 Cycles/byte on Pentium-III. Py is a stream cipher designed especially for very fast and secure encryption. It is intended for use with keys of upto 256 bits (32 bytes) and initial vector (IV) of upto 128 bits (16 bytes) but it can also be used with large keys of

upto 256 bytes and IV sizes upto 64 bytes. The stream generated for a given pair of key and IV is restricted to length of upto 264 bytes.

In the design of Py cipher two rolling arrays have been used. One array P is of 256 bytes that contains a permutation of all the values from 0.... to 255 and second array Y is an array of size 260 where each word is of 32 bit and are indexed as -3,... to 256. Both the arrays rotated in each step of cipher and two output words computed [BS05]. This word is updated by mixing two words of Y into it, where two words are indirectly selected from P and then variable rotation is performed on it. In this way the word is rotated by a number of bits that is calculated from another entry of P.

Attacks on Py

Paul, Preneel and Sekar found a statistical bias in the distribution of the output words that can be used to construct a distinguisher that can work with 283.2 random keys/IVs [PPS06]. In that attack, the key stream can be distinguished from random with 289.2 outputs. Later on Crowley improved this attack by using hidden Markov model by a factor of about 216 in the number of samples [CRO06]. By using this model a distinguishing attack can be made against Py with 272 given bytes of output. 4. Py6Py6 is a variant of Py with reduced internal state size. In this smaller variant of Py, the value of the Permutation P is reduced to a smaller size of 64 and Y to 68 entries. The word size of Y remains unchanged to 32 bits. This variant was proposed to achieve fast initialization with the speed remains the same as the speed of Py. The smaller size of internal states allows a much faster key setup and IV setup that is very attractive for encryption of short streams. This variant has smaller rolling arrays, thus its key setup and IV setup are much faster than of Py, and take 796 and 1464 cycles, respectively. The total number of cycles required by the key and IV setups is thus smaller than the key setup of RC4, and the stream generation is about 2.5 times faster than RC4.Py6 differs from Py in the implementation that in this variant the difference free set had to be replaced by a difference free set modulo 64 and modulo 68. However, there cannot be 10 numbers in such a difference free set, thus in this variant the difference free set contains only the six values that are used as indices to the array P [BS05]. Also, in the generation of the permutation P the internal permutation cannot be used, thus it is removed from one location, and some rotations by eight bits are

replaced by rotations by six bits in order to ensure full mixture of the data. As the indices in this variant are shorter, the length of the generated streams is restricted to 240.

Attacks on Py6

As the design of Py6 is same as the Py the attacks which are applicable to Py are also applicable to Py6. Except those attacks that are applicable to Py, distinguishing attacks were reported against Py6 with 268data and comparable time by Paul and Preneel [PP06].

4.2.1.2 Pypy (Roopy)

Distinguishing attacks were reported against Py by exploiting some statistical bias in the design of Py. To overcome these shortcomings in the design of Py, a new improved version Pypy was proposed by the designers of Py. It takes every second word of the stream of Py (starting from the second word) and half of the outputs are discarded. i.e., the first output of the two outputs at each step is discarded [BS06]. Though, slower than Py, it is still about 1.5 times faster than RC4. Pypy follows the same structure, as of Py in terms of key setup, IV setup and implementation. The only difference being that the first output of the two outputs at each step is discarded

Attacks onPypy

The IV setup of Py and Pypy are same. Wu and Preneel showed that there is serious flaw in the IV setup of Py and Pypy. In these ciphers, two key streams can be identical for every 216 IV's for IV's with special difference. In this way key recovery attacks can be made against the ciphers Py, Pypy, Py6 with chosen IVs [HB06]. This attack was subsequently improved by Isobe et al. They showed that 128 bit key can be recovered with a time complexity of 2^{48} [IOK07]. To overcome the deficiencies in the design of Py, Py6, and Pypy, the designers withdrew them and introduced three new modified or tweaked version of these ciphers that are TPy, Tpy6 &TPypy.

4.2.1.3 TPy, Tpy6 &TPypy

TPy, Tpy6 &TPypy came into existence as a result of major limitations of equivalem IV's found in all members of Py family. TPy, Tpy6 &TPypy are the enhanced

versions of Py, Py6 & Pypy. Since the key setup & stream generation remain unaltered in the new design, both characteristics are same as of the previous version [BS07]. The tweaked IV setup may be very slightly slower than the original, as the modification is very small. The authors of Py claimed that there are no equivalents IVs in Py. When looking at the IV setup, and considering the two loops that mix the IV into EIV (on which the new attack is based), it becomes clear that the authors of Py, when tried to improve the IV setup, added the second loop in order to mix the IV better into the internal state. However, this extra mixing was a poor choice, as if only the first loop was proposed, the IV setup would not have the equivalent IVs problem. To overcome this problem the IV setup of all the three members of the Py family were tweaked. In the tweak of Py and Pypy, the second mixing loop of the IV were modified such that it will not lead to equivalent IVs, by ensuring that it is invertible (following the suggestion of [PPS06]), in a way that prefer over removing this second loop. The final loop of the IV setup was changed slightly, also in order to ensure invertibility. In TPy6, unlike in Py6, the EIV rolling array is also doubled in size to iv sizeb*2. As a result, it limits the keys of TPy6 to be up to 64 bytes, and the IVs to be up to 32 bytes (both are larger than the minimal and recommended sizes). [BS07]

Attacks on TPy, Tpy6 & TPypy

In [SPP071] Sekar, Paul and Preneel published distinguishing attacks on Py, Pypy, TPy and TPypy with data complexities 2^{281} each. In [SPP072] Sekar, Paul and Preneel showed new weaknesses in the stream ciphers TPy and Py. Exploiting these weaknesses distinguishing attacks on the ciphers are constructed where the best distinguisher requires $2^{268.6}$ data and comparable time. In [SPP073] Sekar, Paul and Preneel mounted distinguishing attacks on TPy6 and Py6 with $2^{224.6}$ data and comparable time each. Further in [SPP074] Sekar, Paul and Preneel detected related-key weaknesses in the Py family of ciphers including the strongest member TPypy. Under related keys they shown that a distinguishing attack on TPypy with data complexity $2^{193.7}$ which is lower than the previous best known attack on the cipher by a factor of 2^{88}. It was also shown in this paper that the above attack also works on the other members TPy, Pypyand Py. Later in [BS08], the designers of Py Cipher have denied the attacks on TPy family especially on TPypy.

4.2.1.4 RCR-32 & RCR-64

In the process of getting a secure cipher Paul, Preneel & Sekar proposed two new ciphers RCR-32 and RCR-64 that are derived from TPypy & TPy respectively. The key and IV setup of the RCR-32 and RCR-64 are identical with TPy and TPypy. The only change in the design of these ciphers have been made that variable rotation of quantity s is replaced with constant rotation [SPPD74].

Attacks on RCR-32 & RCR-64

These new variants were claimed to be secure from all attacks that have been made against previous version of the Py family. Any attack has not been reported against these ciphers till date.

Table 4.4 gives a comparative study of encryption speeds of all variants of Py stream cipher and Table 4.5 presents the complexities of attacks on various variants of Py stream cipher.

Table 4.4: Speed of Py and its variant Ciphers in cycles/bytes on Intel Pentium-III processor

CIPHER	SPEED
Py & TPy	2.80
Py6 & TPy6	2.80
Pypy & TPypy	4.58
RCR-32	4.45
RCR-64	2.70
RC4	7.30

Table 4.5: Attacks on the Py-family of stream Ciphers

(It shows the complexity of attacks and X denotes that the attack does not work)

Attacks	Py6	Py	Pypy	TPy6	TPy	TPypy
Paul et al.	X	$2^{89.2}$	X	X	$2^{89.2}$	X
Crowley	X	2^{72}	X	X	2^{72}	X
Preneel-Paul	$2^{68.6}$	X	X	$2^{68.6}$	X	X
Isobe et al.	X	2^{24}	2^{24}	X	X	X

Sekar et al.	X	2^{281}	2^{281}	X	2^{281}	2^{281}
Sekar et al.	X	$2^{268.6}$	X	X	$2^{268.6}$	X
Sekar et al.	$2^{224.6}$	X	X	$2^{224.6}$	X	X
Sekar et al.	X	$2^{193.7}$	$2^{193.7}$	X	$2^{193.7}$	$2^{193.7}$
Preneel -Wu	X	2^{24}	2^{24}	X	X	X

Though Py and its variants used a promising approach that can deliver high performance, analysis have proved susceptibility of these ciphers towards different attacks like distinguishing attack, key recovery attack and chosen IV attack. TPypy is considered to be strongest candidate over the entire family. Even though security claims have been made by Preneel et al [SP073] and Sekar et al. [SP075] against TPypy but the designers have already specified that this cipher can only be assumed secure for 2^{64} bytes of key stream and hence counter claim by designers to prove it secure is also satisfactory [BS08]. No other attack which is more powerful than these attacks and of less complexity has been reported against TPypy till date. Therefore the TPypy can be treated as secure cipher till any other claim or cryptanalysis comes against it.

Analysis of Nonlinear feedback shift register (NLFSR) based Stream ciphers

This Section of the chapter gives a detailed discussion of various NLFSR based stream ciphers and presents a comparative analysis of these ciphers. Different ciphers analysed include DRAGON stream cipher, NLSv2 and GRAIN-128 stream ciphers.

4.3 DRAGON stream cipher

K. Chen, M. Henricksen, W. Millan, J. Fuller, L. Simpson, E. Dawson, H. Lee and S. Moon developed a 32-bit word based stream cipher called Dragon [CHE04]. It is based on a non-linear feedback shift register in conjunction with a non-linear filter function with memory to produce keystream in blocks of 64 bits. Dragon can be executed either with a key and IV-size of 128 bits or a key and IV-size of 256 bits. The nonlinear state update function, which is called the function F, takes six state words (192 bits) as input and produces six words (192 bits) as output. Among the

output words of *F* function, two words are used as new state words and two words are produced as a keystream.

4.3.1 Specification of Dragon

Dragon stream cipher is constructed using a single word based 1024-bit nonlinear feedback register (NLFSR), a nonlinear state update function, and a 64-bit internal memory. Dragon is implemented with two sizes of key and initialization vector (IV) of the size of 128 or 256 bits and produces a 64-bit output per round. The nonlinear state update function *F*, takes six 32-bit words (192 bits) as input and produces six 32-bit words (192 bits) as output. Figure 4.3 gives a detailed structure of function *F* and denotes the input words as {a, b, c, d, e, f} and the output words as {a', b', c', d', e', f'}. Among the output words of *F* function, two words are used as new state words and two words are produced as a keystream. The function *F* has two component functions *G* and *H* constructed by using two explicitly non-linear 8 × 32 S-boxes, which are called as S1 and S2. Suppose that the 32-bit input x is split into four bytes such as x = x0||x1||x2||x3 where xi denotes a single byte and || denotes a concatenation. Each byte xi is passed through an 8 x 32 S-box and the four 32-bit outputs are combined using binary addition.

A network of modular and binary additions is used for diffusion in the F function. It can be divided to three parts: pre-mixing, substitution, and post-mixing. In Figure 4.3. $\ominus$ represents exclusive-or (XOR) and $\boxplus$ represents addition modulo 2^{32}.

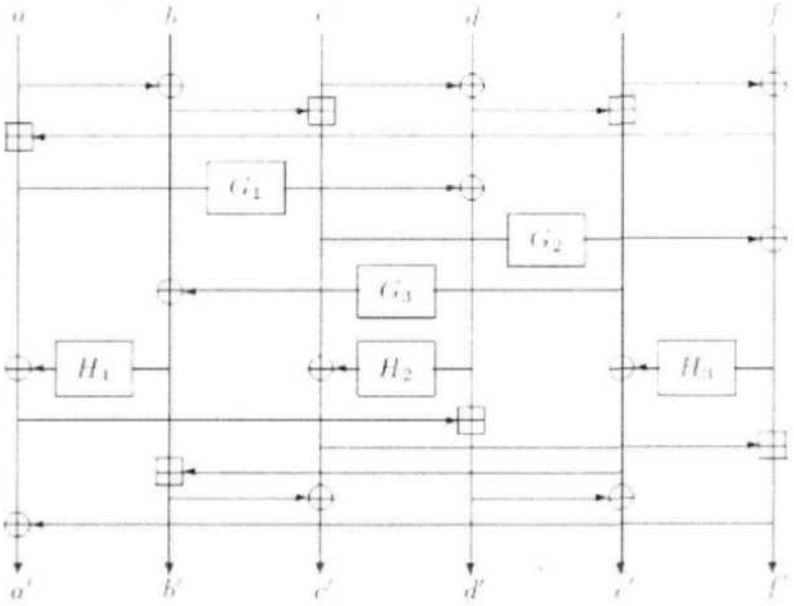


Fig 4.3: Key stream Initialisation and generation

Dragon can be used with two different key and initialisation vector lengths: 128-bit and 256-bit. The 256-bit key and initialisation vector is denoted as K and IV respectively and the 128-bit key and initialisation vector is denoted as k and iv respectively. Dragon has a simple keying (and rekeying) strategy using the key and the publicly known initialisation vector. The 1024-bit internal state is divided into eight 128-bit words, labelled W0 to W7and is initially filled by concatenating the key and the initialisation vector with their bitwise sum and its complement such that W= K || K$\oplus$IV || $\overline{\text{K} \oplus \text{IV}}$ || IV. The state initialisation process makes extensive use of the F function and involves 16 iterations of the F function. The use of existing components for both initialisation and keystream generation simplifies analysis and increases implementation efficiency.

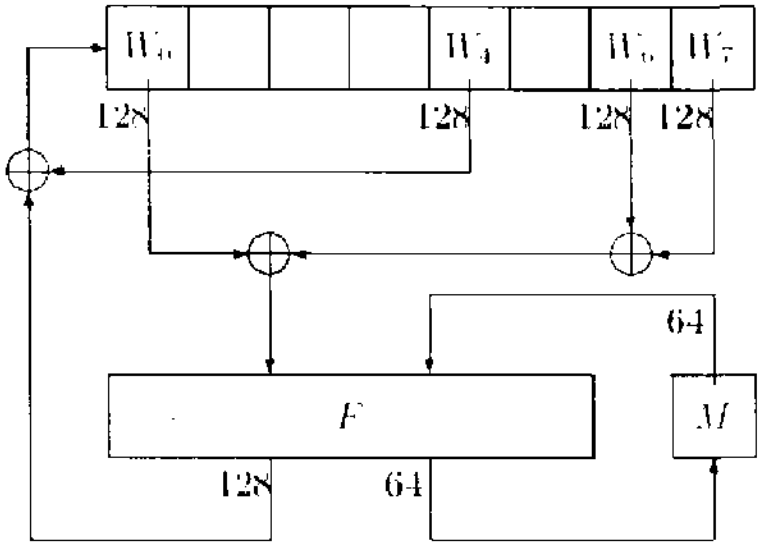


Fig 4.4: Initialisation of Dragon

Dragon possesses a NLFSR of 1024 bit size, which is divided into thirty-two 32-bit values, B_i, $0 < i < 31$. At each round six words from the internal state are used as input to the F function. The words of the first round are taken from the positions 0, 9,16, 19, 30 and 31, which form a Full Positive Differenee Set (FPDS)[HEN05]. Furthermore, the 64-bit memory counter M acts as a counter during keystream generation. The counter M is initialized during the initialization process. Each round of the keystream generation results in output of 64 bits of keystream called k and B and M are updated.

4.3.2 Security of Dragon

Englund and Maximov in [EM05] describe a distinguishing attack on Dragon, which is supposed to be the first proposed attack on Dragon stream cipher. According to Englund and Maximov, Two versions of a distinguishing attack on Dragon were found. The first scenarins requires a computational complexity of $O(2^{187})$ and needs memory only $O(2^{32})$. However, the second scenario has a lower time complexity around $O(2^{155})$, but requires a larger amount of memory $O(2^{96})$. These attacks show that Dragon does not provide full security and can successfully be broken much faster than the exhaustive search, when a key of 256 bits is used.

In response to this attack, the designers of Dragon in [DAW12] state that the attack mentioned in [EM05] is only possible if the designers' restriction on keystream length is relaxed to allow the generation of enormous amounts of keystream under a single key-IV pair. The amount of keystream required is 2^{97} times the 2^{64} bit maximum recommended.

Dragon is believed to possess good statistical properties as it has passed all pertinent statistical tests provided by CRYPT-X[HDLW97]. Based on the size of its NLFSR and COUNTER, Dragon is expected to have a have a period of about 2^{582} bits. Also Dragon is designed to avoid weak keys. The internal state is a non linear feedback shift register that avoids fixed point through its use of a counter. Therefore all-zero state, which is a problematic in many LFSR based stream ciphers, does not produce any weak keys.

4.4 Grain 128:

Specific cryptographic primitives were considered essential for hardware based applications where resources are limited. eSTREAM called for submissions in two categories, hardware and software. Grain 128 was specifically designed for eSTREAM [RM08] submission in hardware profile and it was selected in the final phase of eSTREAM competition.

Grain V0 [HJM05] was originally designed for the eSTREAM submission but some weaknesses were found in the design. A distinguishing attack using the concept linear sequential circuit approximation was reported by Khazaei, Hassanzadeh and Kiaei [KSH05]. Barbein, Gilbert and Maximov [BGM06] also mounted a key recovery attack. In view of these two attacks, the original design slightly modified and feedback polynomial of NFSR and keystream function was modified and designers of Grain presented the new Grain cipher as Grain V1.

But even after these modifications, the design was attacked by Canniere, Kucuk and Preneel [CKP08] by exploiting the weakness in initialization algorithm. This attack was further strengthened by Lee et al. [LY08] by exploiting the same weakness of related key and proposed a key recovery attack against Grain V1. A new attack, called

Dynamic Cube attack [DIS112] has been also reported against the Grain V1 that can recover 80 bits key fully.

In view of these attacks and due to rapid advancement of hardware technology, the possibility of exhaustive key search attack on 80 bit primitives in the near future, a new version that uses 128 bit key called Grain 128 was introduced by the designers of Grain.

4.4.1 Design Specifications of Grain 128:

The basic building blocks of all the variants of Grain stream cipher are same. Grain is a bit oriented synchronous stream cipher. It uses one NFSR and one LFSR with a filter function to combine the output of these shift registers. The general Structure of Grain has been shown in figure 4.5.

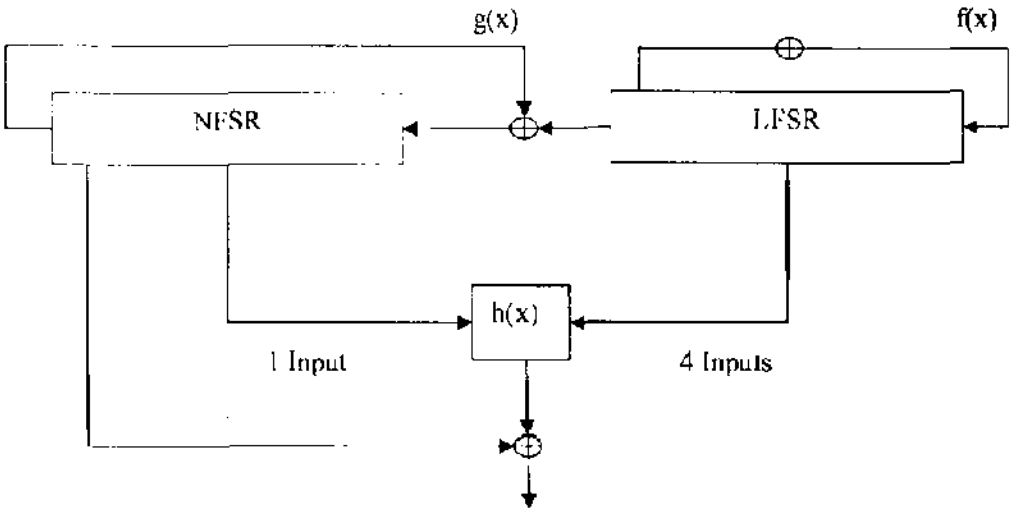


Figure 4.5 : General structure of Grain Stream

In case of Grain 128, NFSR and LFSR are of 128 bit length each has been used. These two registers provide an internal state of 256 bits.

Feedback polynomial of LFSR

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}$$

Feedback polynomial of NFSR

$g(x) = 1 + x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} + x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117}$

The filter function is

$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$

Where the variables x_0 to x_8 respectively correspond to the tap position b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79} and s_{i+95}.

The keystream function is defined as :

$z_i = \sum_{j \in A}^{n} b_{i+j} + h(x) + s_{i+93}$

Where $A = \{2, 15, 36, 45, 64, 73, 89\}$

4.4.2 Key Initialization of Grain:

The Grain cipher is initialized before generating the key streams. The 128 bit secret key is filed in the NFSR and first 96 bits of LFSR are loaded with the IV [HM08]. The remaining bits of LFSR are padded with all ones. The cipher is clocked 256 times before actually generating the keystream. The clocking flush out all the values previously stored in the shift registers and only random values are stored in the two shift registers. The key initialization of Grain has been shown in figure 4.6.

Some weakness in key initialization of Grain 128 was found by Kucuk [KUC06] and in view of these observations, the padding by all ones was changed and only last 31 bits of LFSR were filled with ones and rightmost bit was filled with zero to overcome this weakness.

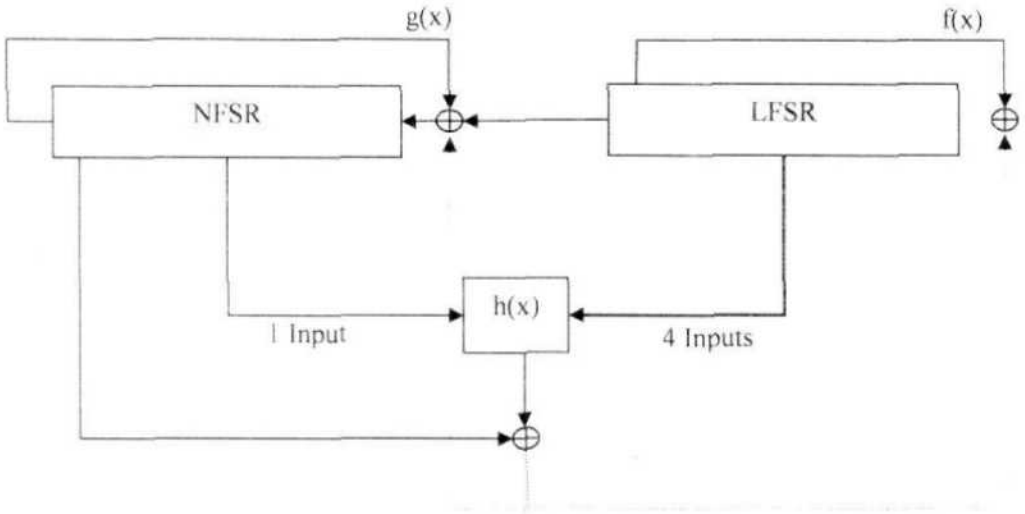


Figure 4.6: Key Initialization of Grain

4.4.3 Attacks on Grain 128:

The design principles of Grain V1 and Grain 128 are same except that Grain V1 is a 80 bit cipher and grain 128 is a 128 bit stream cipher. The attacks which are applicable to Grain V1 are also applicable for grain 128 due to this similarity in design.

Lee et al. [LY08] proposed a key recovery attack that requires 226.59 chosen IVs, 231.39 keystream bits and 227.01 computations to recover the whole key of 128 bits. A fault attack that induces 24 consecutive faults in the LFSR to recover the 128 key within a span of a few minutes was proposed by Berzati et al. [BA09]. A fault attack based on NFSR was proposed by Karmakar and Chowdhury [KSC11] that requires a number of faults ranging from 56 to 256 in NFSR and requires a time complexity of O (2^{21}) and space complexity of O (2^{22}) to recover the secret key. The dynamic Cube attack was also proposed against Grain 128 by Dinur and Shamir [DIS111] that can reduce the key recovery time in comparison to exhaustive key search attack by a factor of 2^{28}.

4.5 NLSV2:

$NLSV_2$ [HP07] is the most advanced member of the SOBER family of stream cipher. It is a modified and advanced version of Non Linear Sober (NLS). NLS was

submitted to eSTREAM in both the profiles 1 and 2 but two major attacks were reported against this design. A linear distinguishing attack was proposed by Chou and Pieprzyk that was based on a linear approximation of NFSR and NLF. The second attack called crossword puzzle attack was also proposed by Cho and Pieprzyk. It was an improved attack based on the first attack. In view of these attacks and suggestions, an improved version of NLS [HP06] called $NLSV_2$ was presented by the designers of NLS. The *Konst* variable is periodically updated in comparison to NLS where *Konst* remain constant during keystream generation and dependent on the key.

$NLSV_2$ is a software oriented synchronous stream cipher that also provides message authentication functionality and it was submitted in eSTREAM competition in profile 1A category. $NLSV_2$ provides 128 bit security and performs 32 bit operations similar to NLS.

4.5.1 DESIGN SPECIFICATIONS OF NLSv2

$NLSV_2$ is very much similar to previous versions of SOBER family ciphers SOBER 128 and NLS except some changes but the design rationale is similar. In this section, the design specification of $NLSV_2$ has been discussed:

4.5.2 Keystream Generation:

The key stream is generated with the help of three components namely nonlinear feedback shift register (NLFSR), a nonlinear filter (NLF) and a counter. The cipher is based on 32 bit operations and 32 bit sized words.

At any time, the state of the NFSR is denoted by $S_t = (r_t[0], r_t[1] \ldots\ldots\ldots, r_t[16])$ and its initial state is represented by $S_0 = (r_0[0], r_0[1] \ldots\ldots\ldots, r_0[16])$.

The update function of NFSR is represented by these equations:

1. $r_{t+1}[i] = r_t[i+1]$, for i=0,1, ...,15
2. $r_{t+1}[16] = f((r_t[0]<<<19) + (r_t[15]<<<9) + Konst) \oplus r_t[4]$
3. if $t \equiv 0$ (modulo $f16$), then
    a. $r_{t+1}[2]$ is modified by adding t (modulo 2^{32})
    b. Konst is changed to the output of NLF

c. The output of NLF at t=0 is not used as a keystream word, where *f*16 is a constant integer $2^{16} + 1 = 65537$

The function f is defined as $f(w)$ = S-Box $(w_{(H)})$ $\oplus$ w where $w_{(H)}$ is the most significant 8 bits of 32 bit word w.

The output of the NFSR is fed to NLF to produce 32 bit output word denoted as V_t.

The NLF generates the output words as:

$$V_t = NLF(S_t) = (r_t[0] + r_t[16]) \oplus (r_t[1] + r_t[13]) \oplus (r_t[6] + Konst)$$

4.5.3 ATTACKS ON $NLSV_2$:

Cho and Pieprzyk [CP04] proposed Crossword Puzzle Attack on NLSv2 that requires 2^{74} observations to mount the distinguishing attack by using a bias in the NFSR and combining with correlation between 29 and 30 bits of S-Box. This distinguisher fails the claim of designers that at least 2^{80} observations are required to mount a distinguishing attack.

In Table 4.6, a comparative analysis of the performance of NLFSR based stream ciphers is presented. These values were calculated on a Pentium M machine with a processor speed of 1700MHZ and variant input values.

TABLE 4.6: Performance Analysis of NLFSR Stream Ciphers

Cipher Primitive	Key	IV	Keystream setup	Keystream encryption (40 Bytes)	Keystream encryption (576 Bytes)	Memory Requirements (in bytes)	Throughput
NLSV2	128	128	777.31	103.51	180.59	2,196	5.59
DRAGON	128	128	113.98	83.20	32.35	4,994	11.74
GRAIN-128	128	96	18.01	59.54	33.67	256*	28.58

(* denotes hardware based stream cipher with a gate count)

In this chapter, an analysis of existing stream ciphers introduced in high profile stream cipher events like eSTREAM and NESSIE is presented. This chapter discusses at length the members of SOBER family of stream ciphers and performed an analysis on the basis of various parameters like byte order considerations, memory requirements, key strength, polynomials used, LFSR over GF (2^w), and security issues. Next an analysis of Py family of stream ciphers is presented, which include Py stream cipher and it tweaked variations. The tweaked versions were essentially introduced once the basic design was found susceptible to various cryptanalytic attacks. The analysis is performed on the parameters like speed and complexities of various attacks on different variants of the Py stream cipher. The second half of this chapter is dedicated to stream ciphers based on Nonlinear feedback shift register. Different NLFSR based stream ciphers discussed in this section include DRAGON, NLSv2 and GRAIN-128. Dragon has been especially built for 32-bit architectures, also bearing in mind the importance of a strong and agile key schedule. The cipher uses the minimum number of operations to ensure security, yet maintains a high level of efficiency. Dragon is believed to provide considerable level of security in terms of cryptanalytic attacks that have been reported on it. NLSv2 is the nonlinear version of the SOBER family of stream ciphers and provides the additional functionality of Message Authentication capability (MAC). Towards the end of the chapter, GRAIN-128 stream cipher is analysed.

BOKHARI: A new software oriented stream cipher

Chapter 5

BOKHARI:
A New Software Oriented Stream Cipher

Traditionally the realm of stream cipher design has focused on linear feedback shift registers (LFSRs), which are well studied and satisfy common statistical criteria but due to their inherent linearity, they become susceptible to several cryptanalytic attacks. The new stream cipher designs are more inclined towards the Non linear feedback functions used in conjunction with Non-linear filter, Non-linear combination generator or irregular clocking to instill nonlinearity in the generated keystream.

The BOKHARI stream cipher is a new proposed stream cipher dedicated to software applications and is designed for a secret key that is up to 128 bits in length. The design is inspired by the design of both SOBER-t16 [HR12] which was one of the four candidates to make to phase II of NESSIE program and a reduced version of the well-known stream cipher DRAGON [CHE04]. BOKHARI is constructed from a *non-linear feedback shift register* (NLFSR) and a *Non Linear Filter* (NLF) and

employs two functions *f1* and *f2* for the initialization and generation of key stream. Figure 5.1 shows a graphical representation of the structure of BOKHARI stream cipher.

Changeover from SOBER-t16

1. The BOKHARI stream cipher is based on a nonlinear feedback shift register (NLFSR) instead of an LFSR used in SOBER-t16
2. Since the support for odd-length keys and initialisation vectors significantly complicate the code [HPR], BOKHARI is designed to only support keys and IVs that are multiples of 4 bytes in contrast to SOBER-t16, which uses multiples of 2 bytes. Keys and IVs of odd-length can always be padded by the user.
3. "Konst" is set to a non-zero value during key loading, so as to avoid poor diffusion.
4. Stuttering has been removed. Thnugh it added to the security but was computationally expensive and exposed cipher to side channel attacks [HPR].

5.1 Design Specifications

BOKHARI is a software oriented stream cipher constructed using an NLFSR, a Non-linear filter and two state update functions *f1* and *f2*. In addition BOKHARI makes use of an S-Box designed by ISRC. Following is a detailed description of various components of BOKHARI stream cipher

5.1.1 State Update Function

BOKHARI uses two state update functions *f1* and *f2*. The *f1* function is a modified version of well-known stream cipher DRAGON and f2 function is similar to that used in SOBER-t16. The f1 function is used in key setup and is a reversible mapping of 64 bits (four 16-bit words) to 64 bits. It takes four 16-bit words as input and produces four 16-bit words as output. In Table 1 the input words are denoted {a, b, c, d} and the output words {a', b', c', d'}. The *f1* function has four component functions denoted G1, G2, G3, G4 as described below. The G function provides algebraic completeness [KD79] and high non-linearity.

A network of modular and binary additions is used for diffusion in the f1 function. It can be divided to three parts: pre-mixing, substitution, and post-mixing.

TABLE 5.1 : Structure of the *f*1 function

Input {a, b, c, d}

Pre-mixing Layer:

1. $b = b \oplus c$; $d = d \oplus a$;
2. $a = a \boxplus d$; $c = c \boxplus b$;

S-box Layer:

3. $d = d \oplus G1(c)$; $b = b \oplus G2(a)$;
4. $a = a \oplus G3(d)$; $c = c \oplus G4(b)$;

Post-mixing Layer:

5. $b' = b \boxplus c$; $d' = d \boxplus a$;
6. $a' = a \oplus d$; $c' = c \oplus b$;

Output $\{a', b', c', d'\}$

5.1.2 G Function

The G function is based on an 8x16 bit s-box which is a combination of skipjack s-box and an s-box tailor designed by the information security Research Centre (ISRC) at the Queensland University of Technology [HR12]. The ISRC S-box was constructed as eight mutually uncorrelated, balanced and highly non-linear single bit function. *aH* and *aL* to form virtual 16×16 s-boxes. The G function consists of four components G_1, G_2, G_3 and G_4 as defined below. The 16-bit input is broken into two bytes (x = x0||x1). Each byte is passed through an 8 × 16 s-box and the two 16-bit outputs combined using binary addition.

G function is defined as:

$$G_1(x) = aH(x_0) \oplus aL(x_1)$$

$$G_2(x) = aL(x_0) \oplus aH(x_1)$$

$$(x) = aH(x_0) \oplus aL(x_1)$$

$$(x) = aL(x_0) \oplus aH(x_1)$$

The *f*2 function is a part of Non Linear Filter (NLF) of BOKHARI. The *f*2 function has a 16 bit input word x = (xH||xL). Here xH and xL represent the upper and lower bytes of 16-bit input word and "||" denotes the concatenation of blocks. The s-box used is same as the one used in *f*1

5.1.3 Initialization

BOKHARI is keyed using a secret key and *t*-byte session key. The internal state is initially filled by the first 17 Fibonacci numbers by setting $r_0 = 1$ and $r_2 = 1$ and computing $r_i = r_{i-1} + r_{i-2}$, for $2 <= i <= 16$. Fibonacci numbers are used only because of the ease of generation. The value of *KONST* is initially set to 0x6996 called INITKONST. The state initialization makes extensive use of the *f*1 function.

5.1.3.1 NLFSR

BOKHARI uses an NLFSR of length 17 with each register element containing a 16-bit word. The state of the NLFSR at time *t* is denoted $s_t,\ldots s_{t+16}$. The new word is generated by shifting the previous state one step and updating the state of the most significant word according to the following equation

$$S_{t+17} = f1\ (s_{t+15} \oplus s_{t+4} \oplus s_t)$$

The register is updated as follows:

$$R[0] = R[1];\ R[1] = R[2];\ldots;R[15] = R[16];\ R[16] = S_{t+17};$$

The *f*1 function used in NLFSR is the feedback function that takes four 16-bit words as input and produces four 16-bit words as output. In Table 5.1 the input words are denoted {a, b, c, d} and the output words denoted as {*a'*, *b'*, *c'*, *d'*}. The input word *a*, *b*, *c*, *d* are assigned the values at *Tap position* {0, 4, 15, 17} of the NLFSR

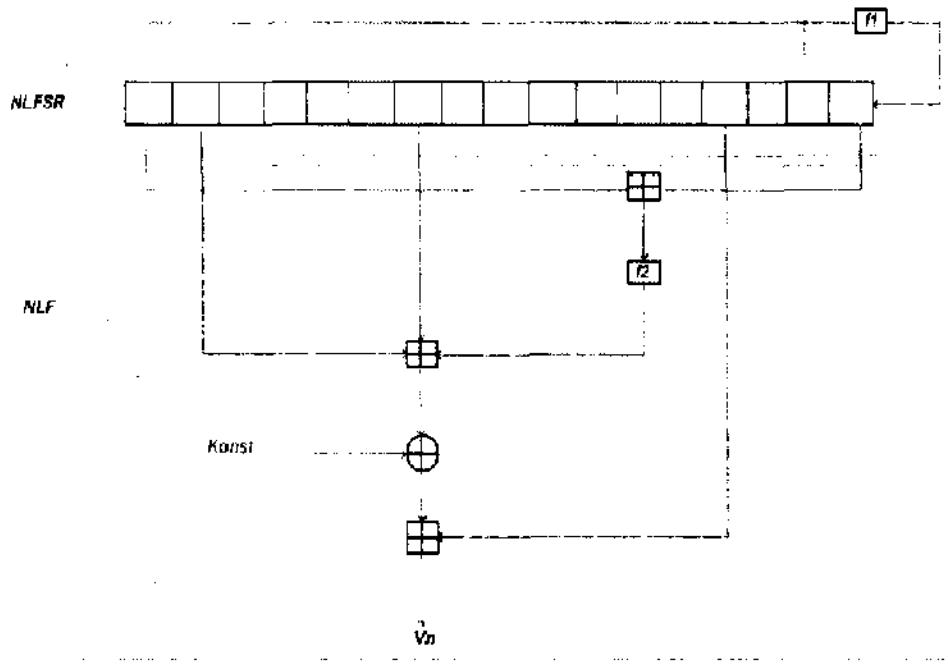


Figure 5.1: Graphical Representation of BOKHARI Stream Cipher

5.1.3.2 The Non-linear filter (NLF)

At time t, the nonlinear filter (NLF) takes five words s_t, s_{t+1}, s_{t+6}, s_{t+13}, s_{t+16} from the NLFSR states and the constant *Konst*as the input and produces the output v_t. The nonlinear filter consists of function $f2$, three adders and XOR addition. The function $f2$ translates a single word input into a single word output. The output is generated by following equation

$$v_t = ((f(s_t \boxplus s_{t+16}) \boxplus s_{t+1} \boxplus s_{t+6}) \oplus Konst) \boxplus s_{t+13}$$

The function $f2$ uses an S-box with an 8 bit input and a 16 bit output. The S-box which is a combination of skipjack s-box and an s-box tailor designed by the information security Research Centre (ISRC) at the Queensland University of Technology [HR12].The input x to the function $f2(x)$ is split into two strings, xH and xL.

5.1.3.3 The S-Box

The S-Box used in BOKHARI stream cipher is a combination of the Skipjack S-box and an S-box tailor designed by Information Security Research Center (ISRC) at Queensland University of Technology (QUT) and is employed in functions $f1$ and $f2$.

In case of function $f1$, the 16 bit input 'a' is divided into two parts aH and aL.SBox is fed with an input aH. The eight most significant bits (MSBs) of the output of the SBox are equal to the output of the Skipjack S-box and the eight least significant bits (LSBs) of the output of the SBox are equal to the output of the S-box constructed by the ISRC. These two 16 bit outputs are then XORed with each other and fed back to function G.

In case of function $f2$ used in the NLF part of the cipher, the 16 bit input is processed in the similar fashion as in $f1$. The only difference is that the eight MSBs of the output of $f2$ is the output of the skipjack S-box, while the eight LSBs are obtained by XORing the eight bits of the output of the ISRC S-box with the LSBs of the input.

5.1.4 Key stream generation

Principle operation employed for keying and rekeying of BOKHARI are:

Include (X): This operation adds the word X to r_{15} modulo 2^{16}

Diffuse (): This operation clocks the register, obtains the output v of the NLF and replaces the value of r[4]with the value of (r[4]$\oplus$ v).

All keys must be multiple of 4 bytes in length; keylen is the length of the key in bytes. Eight 16-bit variables $a, b, c, d, a', b', c', d'$ act temporary placeholders for register contents and outputs of function $f1$. This algorithm uses the values in "k []" to transform the current status of the register.

Input = {k [], keylen}

1. Convert "k[]" into kwl =keylen/4 words and store in an array "kw[]" of kwl "big-endain" words:
2. For each i, $0 \leq i \leq (kwl-1)$, Include (kw[i]) and apply Diffuse ().
3. $a=r_0$; $b=r_4$; $c=r_{15}$; $d=r_0 \oplus r_4 \oplus r_{15}$.
4. $\{a', b', c', d'\} = f1$ (a, b, c, d).

5. $r_{16} = a' \oplus b' \oplus c' \oplus d'$.
6. Apply Diffuse () 17 more times

The register is initialized to first 17 Fibonacci numbers.

$r_0 = r_1 = 1$; $r_i = r_{i-1} + r_{i-2}$ for $2 < i < 16$.

And value of constant is set to the word 0x6996 (called INITKONST) in contrast to SOBER-t16 that uses an all zero word as the initial value of Konst. The cipher applies Loadkey (K[], t), which includes the session key bytes into the register and diffuses the information throughout the register. Again in BOKHARI, *keylen* is not included in the register as used in SOBER-t16; instead the function f1 is used to ensure strong nonlinearity and distinct initial states. Once the key is diffused, the register is clocked and value of NLF is calculated. Konst is set to this new value. The values at NLFSR tap sets are stored in temporary placeholders {a, b, c and d} and passed to the function *f*1 that calculates the output by performing sufficient transformations of the values stored in these variables. The Diffuse () operation is applied 17 times in order to ensure that every bit of inputs effects every bit of resulting register state in a nonlinear fashion. If the cipher is going to be rekeyed for each of multiple frames, then the 17 word state of register (which is called the initial key state) can be saved at this point for later use and the key discarded. However for shorter keys, the key could be saved and the keying procedure repeated as necessary, trading additional computation time for some extra memory. If the cipher is not being used with a fame key, then the cipher pruduces a key stream with the register starting in the initial key state. However, if the cipher uses a frame key, then the cipher first resets the register state to the saved initial key state and loads the *n*-byte frame key *frame*[0], ..., *frame*[*n*-1] using Input (*frame*[], *n*). The state of the register following the re-keying is taken as the initial state and the cipher produces a key stream with the register starting in this state.

5.2 Security Claims

Any attack on BOKHARI stream cipher is believed to have a complexity greater than an exhaustive key search, though no mathematical proof of security is claimed. A brief Analysis of several attacks on BOKHARI stream cipher is mentioned below

5.2.1 Correlation Attack

The feedback function of the stream cipher is highly nonlinear through the use of SBox and a network mesh of addition and XOR preceded and followed by some pre-mixing and post-mixing. The nonlinearity of the feedback function and the selection of taps for the output filter function should adequately disguise any short distance correlations. The regular modification of the state should enhance this effect for long term correlations.

5.2.2 Guess and determine attacks

The existence of Guess and Determine attack relies on the tap sets rather than individual NLF and NLFSR operations. [HR03] The taps selected for BOKHARI were such that they could resist Guess and Determine attacks. The tap set used in NLFSR state update function {0, 4, 15, and 17} and Nonlinear filter {0, 1, 6, 13, 16} form a full positive difference set (FPDS) that prevents Guess and Determine attacks.

5.2.3 Distinguishing Attacks

If the output sequence of a stream cipher can be statistically distinguished from a random sequence, then the cipher is not strong enough for cryptographic applications. BOKHARI is designed with complex initialisation and update function. It has no linear masking and therefore immune to this type of distinguishing attacks. [CHJ02]

5.3 Weak Keys

Weak keys are those keys that bypass some operations of the cipher. That is, the operations have no effect in the calculation of the feedback or the output keystream. BOKHARI is designed to avoid weak keys. Since the internal state is an NLFSR and value of *KONST* is set to non zero, therefore the all zero state is not is avoided. While it is easy to bypass the pre-mixing phase in a single iteration of the F function by having repetitive inputs such as all zeros or all ones, it is only possible for the first of the 17 iterations of F in the key setup. Also, selected values are limited to the first four inputs of the F function, as the feedback input takes the value of final output v_n. The network of G and H functions ensure that the initial states which bypass the pre-mixing phase cannot bypass any other operations in F. it is believed that the above design features provide a strong guarantee that there are no weak keys for Dragon

5.4 Statistical Analysis

Key streams generated by BOKHARI stream cipher have been analysed using NIST statistical suite [KD79]. The NIST test battery consists of 15 statistical tests. These tests check the input sequence for randomness, linear complexity, patterns and several other statistical properties exploited in crypt- analytical attacks. No bias was found by any of the 17 tests included in the NIST suite. The key stream generated by BOKHARI is reported to have properties of random bit streams and sufficient tolerance pertinent to these attacks

TABLE 5.2 : Statistical Analysis

Statistical test	P-value (BOKHARI)
Frequency	0.064721
Block frequency	0.515168
Cumulative sum- Forward	0.371755
Cumulative sum- Backward	0.307124
Runs	0.183759
Long Runs of Ones (M=1000)	0.447910
Rank	0.658201
Spectral DFT	0.894390
Non Overlapping Templates (m=9, b=0000000001)	0.315012
Overlapping Template	0.271304
Universal (L=7, Q=1280)	0.621864
Approximate Entropy (m=10)	0.531830
Random Excursions (x=+1)	0.871063
Random Excursions Variant (x=-1)	0.418520
Linear Complexity (M=500)	0.741940
Serial (m=5)	0.428531

ENT Tests:

BOKHARI Stream Cipher was tested on ENT test suite and the recorded results are given in Table 5.3 and found totally unbiased. The tests were run on an input file containing 10400 samples and the output recorded that the input sequence was totally uncorrelated (-0.038915), entropy was recorded as 1.196567, which is optimum and arithmetic mean value of data byte was found to be 47.0118 which is totally random

Table 5.3: ENT Test Results

Test	Result
Entropy	1.196567
Compression rate <Input file: 10400 bytes>	85%
Chi square distribution <10400 samples>	1224599
Arithmetic mean value	47.0118
Monte Carlo value	4.00000
Serial Correlation coefficient	-0.038915

5.5 Memory requirements:

Storage requirements include 1,024 bytes to store a non-linear SBox containing 256 entries with each entry being a 16 bit word and 1,024 bytes to store the multiplication table with 256 entries, therefore a total of 2,048 bytes to store the internal state. Further storage of 140 bytes is required for the register of length 17 words and a four byte Konst. BOKHARI has memory requirements totaling 2,188 bytes. This is suitable for even very constrained environments.

BOKHARI stream cipher was run on an Intel (R) Core (TM) i7-3770 @3.40 GHz machine and compiled with gcc. The performances were obtained by taking the average of multiple runs.

TABLE 5.4: Performance of BOKHARI on Intel-i7 @3.40 GHz

Primitive	Key	IV	CPU time (secs)	Encryption speed (in secs) (64 bits)	Encryption speed (in secs) (128 bits)	Encryption speed (in secs) (256 bits)	Throughput (Bytes per sec)
BOKHARI	128	128	0.06858	0.006	0.0065	0.007	1334

In this chapter, a new software oriented stream cipher based on a secret key of length 128 bits is presented. This chapter gives an insight of the design details of BOKHARI stream cipher and assessment of its security. It is believed that any attack on BOKHARI has a complexity greater than exhaustive key search. This chapter also provides the statistical analysis of the new cipher and source code of BOKHARI written on a gcc compiler can be found at the end of this thesis as Appendix II.

Conclusion

Chapter 6

Conclusion

In this thesis, the theory and practice of symmetric-key cryptology is discussed. Cryptographic algorithms provide mathematical tools for achieving several important security objectives in electronic communications. Among the most important of these objectives is ensuring privacy of the message exchange. Privacy can be achieved by using both symmetric and asymmetric techniques. Though, in the areas where the highest level of security, high throughput capacity and low implementation costs are of the utmost importance, there is no alternative to symmetric systems and to stream ciphers, in particular.

Stream Ciphers were widely used without any standard design. Now a day's stream ciphers have attracted attention of scientific community and several attempts were made to standardise it but it these are still in nascent stage. For the construction of stream ciphers, the toolbox of good, well documented strategies is not as rich as block ciphers. Most theoretical results are concerned with Boolean combining functions or nonlinear filters.

In this thesis, the basic design structure and philosophy of Stream cipher design is discussed and an analysis on the available stream cipher designs and cryptanalytic attacks on these ciphers is carried out. This whole investigation gives an insight about the state of the art of stream cipher design, which is a pre-requisite for efficient and secure stream cipher design.

The main contribution of this thesis is a proposal of a new stream cipher named a "BOKHARI stream cipher" that has been successfully tested on different statistic tests and with the available knowledge of crypt analytical attacks and it has be found to be secure against any such weakness.

Future Work

This work leaves room for interesting problems for future work. The usage of sh key and IV makes BOKHARI a good candidate to be used in constrained per devices. The integration of Message Authentication Code (MAC) in BOI stream cipher can be a vital enhancement to the existing design. MAC implemented in two phases: MAC accumulation and Mac finalisation.

References

References

[AAR02]	Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P."**The EM Side-Channel(s): Attacks and Assessment Methodologies**". Cryptographic Hardware and Embedded Systems – CHES 2002 (2002)
[AE97]	A. Ahmad and A.M Elabdallai. "**An Efficient Method to Determine Linear Feedback Connections in Shift Registers That Generate Maximal Length Pseudo-Random Up And Down Binary Sequences**". Computer Electronic Engineering Vol.23, No.1 pp. 33-39, 1997
[AH88]	H. R. Amirazizi and M. E. Hellman. "**Time-memory-processor trade-offs**". IEEE Transactions on Information Theory, 34(3):505–512, 1988.
[ARM04]	F. Armknecht, "**Improving fast algebraic attacks**", Fast Software Encryption (FSE) 2004, LNCS 3017, Springer Verlag, pp. 65-82, 2004.
[BA09]	Berzati, Alexandre, et al. "**Fault analysis of GRAIN-128.**" Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on. IEEE, 2009.
[BAB95]	S. Babbage. "**Improved exhaustive search attacks on stream ciphers**". In ECOS 95 (European Convention on Security and Detection), 1995.
[BAM10]	Bokhari, M.U., ShadabAlam, and FaheemSyeedMasoodi. "**Comparative analysis of Py (Roo) family of stream ciphers.**" ICRITO-2010, November 2010, Faidabad (India) pp 310-313
[BCL03]	S. Babbage C. De Cannière J. Lano B. Preneel and J. Vandewalle.**"Cryptanalysis of SOBER-t32.**" Preproceedings of Fast Software Encryption FSE2003 February 1999 pp. 119-136.
[BGM06]	Côme Berbein, Henri Gilbert, and Alexander Maximov. "**Cryptanalysis of grain.**"Fast Software Encryption. Springer Berlin Heidelberg, 2006.
[BL11]	S. Babbage and J. Lano. "**Probabilistic Factors in the Sober-t Stream Ciphers** " third open NESSIE Workshop proceeding 2002International Journal of Computer Applications (0975 – 8887) Volume 23 No.1 June 2011
[BM09]	A. A. Bruen and R. A. Mollin."**Cryptography and Shift Registers**", The Open Mathematics Journal, 2009, pp 16-21.
[BM10]	BokhariM. U.and FaheemMasoodi, "**Comparative Analysis of Structures and Attacks on Various Stream Cipher**", Proceedings of the 4th National Conference: INDIACom-2010. pp. 236 – 238.
[BM12]	Bokhari, M. U., and Faheem Masoodi. "**BOKHARI: A new software oriented stream cipher: A proposal.**" Information and Communication Technologies (WICT), 2012 World Congress on. IEEE, 2012.

[BP99]	D. Bleichenbacher and S. Patel "**SOBER cryptanalysis**" Pre-proceedings of Fast Software Encryption '99 1999 pp. 303-314.
[BS00]	A. Biryukov and A. Shamir. "**Cryptanalytic time/memory/data tradeoffs for stream ciphers**". In T. Okamoto, editor, ASIACRYPT, volume 1976 of Lecture Notes in Computer Science, pages 1–13. Springer, 2000.
[BS05]	E. Biham, J. Seberry, "**Py (Roo): A Fast and Secure Stream Cipher Using Rolling Arrays.**" available at http://www.ecrypt.eu.org/stream/ciphers/py/py.psecrypt submission 2005
[BS06]	E. Biham, J. Seberry, "**Pypy (Roopy): Another version of Py**", ecrypt submission 2006
[BS07]	Eli Biham, Jennifer Seberry, "**Tweaking the IV Setup of the Py Family of Ciphers The Ciphers TPy, TPypy, and TPy6**" , Published on the author's webpage at http://www.cs.technion.ac.il/~biham/, January 25, 2007.
[BS08]	E. Biham, J. Seberry, "**The Truth on TPy**", February 2008, available at http://www.ecrypt.eu.org/stream/papersdir/2008/014.pdf
[BSF11]	Bokahri, Shadab and Faheem. "**A Review of Py (Roo) Stream Cipher and its Variants**". In Proceedings of the 5th National Conference; INDIACom-2011 Computing For Nation Development, March 10 – 11, 2011 Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi
[BSF12]	Bokhari, M. U., Shadab Alam, and Faheem Syeed Masoodi. "**Cryptanalysis Techniques for Stream Cipher: A Survey**." International Journal of Computer Applications 60.9 (2012).
[BSW78]	A. Biryukov, A. Shamir, and D. Wagner. "**Real time cryptanalysis of A5/1 on a pc**". In B. Schneier, editor, FSE, volume 1978 of Lecture Notes in Computer Science, pages 1–18. Springer, 2000.
[CAM10]	Cameron McDonald. "**Analysis of Modern Cryptographic Primitives**" . A thesis submitted to Macquarie University for the degree of Doctor of Philosophy Department of Computing, Faculty of Science January 2010
[CAN01]	Christopher De Canniere "**Guess and Determine Attack on SOBER**" 2001
[CAN05]	Anne Canteaut, "**Fast correlation attacks against stream ciphers and related open problems**". Theory and Practice in Information-Theoretic Security,. IEEE Information Theory Workshop, 2005
[CB86]	Cagigal, N.P; Bracho, S "**Algorithmic Determination of linear feedback in a Shift Register for pseudorandom binary sequence generation**" Electronic Circuits and Systems, IEE Proceedings. 1986 pp. 191 - 194 Vol. 133
[CHE04]	Chen, Kevin et al. "**Dragon: A fast word based stream cipher.**" Information Security and Cryptology–ICISC 2004 (2005): 33-50.
[CHJ02]	Don Coppersmith, S. Halevi and C. Jutla, "**Cryptanalysis of stream ciphers**

	with linear masking" In M. Yung, editor, Advances in Cryptology- CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, proceedings, volume 2442 of Lecture Notes in Computer Science, pages 515-532, Berlin Heidelberg, 2002. Springer-Verlag.
[CKP08]	De Cannière, Christophe, Özgül Küçük, and Bart Preneel. "**Analysis of Grain's initialization algorithm.**" Progress in Cryptology–AFRICACRYPT 2008. Springer Berlin Heidelberg, 2008. 276-289.
[COU02]	N. Courtois, "**Higher Order Correlation Attacks**", XL algorithm and Cryptanalysis of Toyocrypt, ICISC 2002, LNCS 2587, Springer-Verlag, pp. 182-199, 2002.
[COU03]	N. Courtois, "**Fast Algebraic Attack on Stream Ciphers with Linear Feedback**", Advances in Cryptology - Crypto 2003, LNCS 2729, Springer-Verlag, pp. 176-194, 2003.
[CP04]	JOO Yeon Cho and Josef Pieprzyk "**Algebraic Attacks on SOBER –t 32 and SOBER-t16 without stuttering**". Fast Software Encryption-FSE 2004
[CP041]	Joo Yeon Cho and Josef Pieprzyk, "Crossword Puzzle Attack on NLSv2", pdf, submitted 2006-09-04, updated 2006-11-29. Available at http://www.ecrypt.eu.org/stream/allcandidates.html#profile1
[CRO06]	P. Crowley, "**Improved Cryptanalysis of Py**," Workshop Record of SASC 2006 - Stream Ciphers Revisited, ECRYPT Network of Excellence in Cryptology, February 2006, Leuven (Belgium), pp. 52-60 also available at http://www.ecrypt.eu.org/stream/papersdir/2006/010.pdf
[DAW12]	Ed Dawson et. al., "**Dragon is well and Alive**" Available at http://www.ecrypt.eu.org/stream/papersdir/069.pdf Accessed on 14 July, 2012
[DH76]	W. Diffie and M.E. Hellman. "**New directions in cryptography**". IEEE Transactions on Information Theory, 22:644-654, 1976
[DIS11]	Dinur, Itai, and Adi Shamir. "**Breaking Grain-128 with dynamic cube attacks.**"Fast Software Encryption. Springer Berlin Heidelberg, 2011.
[DJS09]	P. P. Deepthi, Deepa Sara John and P. S. Sathidevi"**Design and analysis of a highly secure stream cipher based on linear feedback shift register**", Elsevier, computers and electrical engineering (2009), pp 235-243.
[DKL98]	J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, "**A practical implementation of the timing attack**", Proc. CARDIS 1998, Smart Card Research and Advanced Applications (J.-J. Quisquater and B. Schneier, eds.), LNCS, Springer, 1998.
[DS01]	M. Dichtl and M. Schafheutle "**Linearity Properties of the SOBER-t32 Key Loading**" NESSIE Public Document NES/DOC/SAG/WP5/046/1 November 2001

[DUB09]	Elena Dubrova, "**A Transformation from the Fibonacci to the Galois NLFSRs**", IEEE Transaction on Information Theory, Vol 55, No. 11, NOV 2009, pp 5263-5271
[ECR10]	eSTREAM - The ECRYPT Stream Cipher Project. http://www.ecrypt.eu.org/stream
[EJ00]	Ekdahl, Patrik, and Thomas Johansson. "**SNOW-a new stream cipher.**"Proceedings of First Open NESSIE Workshop, KU-Leuven. 2000.
[EJ02]	Ekdahl, P., Johansson, T., 2002." **Distinguishing attacks on SOBER-t16 and SOBER-t32**. In: Daemen, J., Ri-jmen, V. (Eds.), Fast Software Encryption 2002. Vol. 2365 of Lecture Notes in Computer Science. Springer-Verlag, pp. 210–224.
[EJ02]	PatraikEkdahl and Thomoas Johansson "**Distinguishing attacks on SOBER-t16 and SOBER-t32** " Fast Software Encrytion FSE 2002 Springer-Verlag LNCS 2365 pp. 210-224
[EM05]	Englund and Maximov, "**Attack the Dragon**" Lecture Notes in Computer Science Volume 3797, 2005, pp 130-142
[ENG07]	Englund, Håkan. "**Some Results on Distinguishing Attacks on Stream Ciphers**". Department of Electrical and Information Technology, Lund University, 2007
[ERI11]	Cole, Eric. Network Security Biblevol 786, wiley 2011
[EST12]	eSTREAM committee's "**Short Report on the End of the Second Phase**" available at http://www.ecrypt.eu.org/stream. Accessed on 23 Sep, 2012
[FGK07]	W. Fischer, B. M. Gammel, O. Kniffler, J. Velton, "**Differential Power Analysis of Stream Ciphers**," Topics in Cryptology-CT-RSA 2007, Springer-Verlag, LNCS, Vol. 4377, pp. 257–270, 2007.
[FIN94]	Hal Finney, An RC4 Cycle that Can't Happen, Usenet newsgroup sci. crypt, September 1994.
[FM00]	S. R. Fuhrer and D. A. McGrew "**Statistical analysis of the alleged RC4 stream cipher.**" in Proceedings of Fast Software Encryption {FSE'00 (B. Schneier, ed.), Lecture Notes in Computer Science, Vol. 1978, pp. 19{30, Springer-Verlag, Berlin, 2000.
[FMS01]	Scott Fluhrer, ItsikMantin, and Adi Shamir. "**Weaknesses in the key scheduling algorithm of RC4**". In Mitsuru Matsui, editor, Proceedings of the 8th International Workshop on Fast Software Encryption, volume 2355 of Lecture Notes in Computer Science, pages 1–24. Springer-Verlag, 2001.
[GMO01]	K. Gandolfi, C. Mourtel and F. Olivier. "**Electromagnetic Attacks: Concrete Results**". In the Pro-ceedings of the Workshop on Cryptographic Hardware and Embedded Systems 2001 (CHES 2001), LNCS 2162 Paris, France, May 2001, pp 251–261

[GOL97]	J. D. Golic. "**Cryptanalysis of alleged A5 stream cipher**". In EUROCRYPT, pages 239–255, 1997.
[GOL197]	J.D. Golic, "**Linear statistical weakness of alleged RC4 keystream generator.**" in Proceedings of Eurocrypt'97 (W. Fumy, ed.), Lecture Notes in Computer Science, Vol. 1233, pp. 226{238, Springer-Verlag, Berlin, 1997.
[GOR02]	Mark Goresky: Fibonacci and Galois "**Representation of Feedback-with-carry shift registers**", IEEE Transactions on Information Theory, Vol 48, Nov 2002
[HB06]	Hongjun Wu, Bart Preneel, "**Key Recovery Attack on Py and Pypy with Chosen IVs**", eSTREAM report 2006/052 (2006) URL: http://www.ecrypt.eu.org/stream/papers.html
[HEL85]	Tore Herlestam "**On functions of Linear Shift Register Sequence**" Eurocrypt 85 LNCS 219 F. Pichler Ed. Springer-Verlag pp.119-129 1985.
[HEN05]	Matt Henricksen, "**Design, Implementation and Cryptanalysis of Modern symmetric Ciphers**" 2005
[HEY02]	H. Heys "**A Tutorial on Linear and Differential Cryptanalysis**".2002 Available at: http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf
[HJM05]	M. Hell, T. Jonasson, and W. Meier. "**Grain- A Stream Cipher for Constrained Environments**". ECRYPT Stream Cipher Project Report 2005/001, 2005. Available at http://www.ecrypt.eu.org/stream
[HM08]	Hell, Martin, et al. "**The Grain family of stream ciphers.**" New Stream Cipher Designs. Springer Berlin Heidelberg, 2008. 179-190
[HMP05]	Philip Hawkes Cameron McDonald Michael Paddon Gregory G.Rose and Miriam Wiggers de Vries "**Primitive Specifications for NLSv2**" 2005
[HN00]	J. Hastad and M. NaslundBmgl: "**Synchronous keystream generator with provable security**". Primitive submitted to NESSIE Sep.2000.
[HP06]	Hawkes, Philip, et al. "**Primitive specification for NLSv2.**" See http://www.ecrypt.eu.org/strearn/nlsp3.html (2006).
[HP07]	Hawkes, Philip, et al. "**Primitive specification for NLSv2.**" See http://www.ecrypt.eu.org/strearn/nlsp3.html (2007).
[HR00]	P. Hawkes and G. Rose "**Primitive Specification and Supporting Documentation for sober-t32 submission to NESSIE**" First NESSIE Workshop Proceedings 2000.
[HPR]	Philip Hawkes, Micheal Paddon and G. Rose"**Primitive Specifications of SOBER 128 "** (Version 2) Available at http://eprint.iacr.org/2003/81/ .
[HR02]	P. Hawkes and G. G. Rose. "**Guess-and-Determine Attacks on SNOW**". In Selected Areas in Cryptography, pages 37–46, 2002.
[HR031]	P. Hawkes and G. Rose "**Turing: a fast steam cipher**" First Software Encryption FSE- 2003 pre-proceedings pp. 307-324 2003.

[HR032]	Philip Hawkes and G. Rose"**Primitive Specifications of SOBER 128** " IACR ePrint Archive 2003 Available at http://eprint.iacr.org/2003/81/ .
[HR12]	P. Hawkes and G. Rose, "**Primitive Specification and Supporting Documentation for sober-t16 submission to NESSIE**". NESSIE submission for SOBER-t16,date: 05/23/12
[HS05]	Hong, Jin, and PalashSarkar. "**Rediscovery of time memory tradeoffs.**" IACR ePrint Archive, Report 90 (2005): 2005.
[HVT05]	Henk C.A. van Tilborg"**Encyclopaedia of Cryptography and Security**" Springer. 2005
[HDLW97]	Helen Gustafon, Ed Dawson, Laurel Nielsen and Willaim Ceili, "**A computer package for measuring the strength of stream ciphers**" Journal of Computers and Security, 13(8): 687-697, 1997
[IOK07]	T. Isobe, T. Ohigashi, H. Kuwakado, and M. Morii. "**How to Break Py and Pypy by a ChosenIV-attack**" available at http://www.ecrypt.eu.org/stream/papersdir/2007/035.pdf
[JJD09]	Sun Jing, Yang jing-yu, Fu De-sheng **" Research On the Security of Key Generator in Stream Ciphers"** The 1st International Conference on information Science and engineering (ICISE2009) pp. 1831--834
[KD79]	J. Kam and G. Davida. "**Structured Design of Substitution-Permutation Encryption Networks**". IEEE Transactions on Computers, 28(10):747–753, October 1979.
[KJJ99]	P. Kocher, J. Jaffe, B. Jun, "**Differential Power Analysis"**, in the Proceedings of Crypto 1999, LNCS, vol 1666, pp 398–412, Santa-Barbara, CA, USA, August 1999.
[KMP98]	L.R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege, "**Analysis methods for (alleged) RC4.**" in Proceedings of Asiacrypt'98 (K. Ohta and D. Pei, eds.), Lecture Notes in Computer Science, Vol. 1514, pp. 327{341, Springer-Verlag, Berlin, 1998.
[KOC96]	P. Kocher, "**Timing attacks on implementations of Diffe-Hellman, RSA, DSS, and other systems**", Advances in Cryptology - CRYPTO '96, Sant Barbara, California (N. Koblitz, ed.), LNCS, vol. 1109, Springer, 1996, pp. 104-113.
[KR95]	B. Kaliski and M. Robshaw. "**Message authentication with MD5**". CryptoBytes, 1(1):5-8, spring 1995.
[KSC11]	Karmakar, Sandip, and Dipanwita Roy Chowdhury. "**Fault analysis of grain-128 by targeting NFSR.**" Progress in Cryptology-AFRICACRYPT 2011. Springer Berlin Heidelberg, 2011. 298-315.
[KSH05]	Khazaei, Shahram, Mehdi Hassanzadeh, and Mohammad Kiaei.

	"**Distinguishing attack on grain.**" 2005-12-01)[2009-01-12]. http://www. ecrypt. eu. org/stream/papersdir/071. pdf (2005).
[LJ10]	W. Liang and Long Jing, "**A cryptographic Algorithm Based on Linear Feedback Shift Register**", 2010 International conference on computer application and system Modeling (ICCASM 2010), v15, pp 526-529
[LY08]	Lee, Yuseop, et al. "**Related-key chosen IV attacks on Grain-v1 and Grain-128.**"Information Security and Privacy. Springer Berlin Heidelberg, 2008.
[LZH09]	Gregor Leander, Erik Zenner, and Philip Hawkes"**Cache Timing Analysis of LFSR-based Stream Ciphers**, Proc. Crypto & Coding 2009, Springer LNCS 5921, 2009
[MAB12]	FaheemMasoodi, ShadabAlam and M. U. Bokhari, "**An Analysis of LFSR used in Stream Ciphers**", International Journal of Computer Applications, USA. Volume 46, No 17.
[MAB11]	FaheemMasoodi, Shadab Alam and M U Bokhari **"SOBER Family of Stream Ciphers: A Review"**: International Journal of Computer Applications (2011), Vol. 23.
[MAN01]	I. Mantin, Analysis of the Stream Cipher RC4, M.Sc. thesis, TheWeizmann Institute of Science, 2001. Available athttp://www.wisdom.weizmann.ac.il/_itsik/RC4/rc4.html.
[MAS92]	Massey, James L. Contemporary cryptology**"an introduction. Contemporary Cryptology"** The Science of Information Integrity, GJ Simmons, ed., IEEE Press, 1992.
[MAT06]	J. Mattsson, "**A Guess and Determine Attack on the Stream Cipher Polar Bear**". eSTREAM, ECRYPT Stream Cipher Project, Report 2006/017, 2006. http://www.ecrypt.eu.org/stream.
[MIR02]	I. Mironov, **"(Not so) random shu_es of RC4**." in Proceedings of Crypto'02 (M. Yung, ed.), Lecture Notes in Computer Science, Vol. 2442, pp. 304{319, Springer-Verlag, Berlin, 2002.
[MRO08]	Matthew J. B. Robshaw and Olivier Billet, editors. "**New Stream Cipher Designs- The eSTREAM Finalists**", volume 4986 of Lecture Notes in Computer Science. Springer, 2008.
[MS01]	I. Mantin and A. Shamir, "**A practical attack on broadcast RC4**." in Proceedings of Fast Software Encryption { FSE'01 (M. Matsui, ed.), Lecture Notes in Computer Science, Vol. 2355, pp. 152{164, Springer-Verlag, Berlin, 2001.
[MS88]	W. Meier and O.Staffelbach, "**Fast Correlation Attacks on Stream Ciphers**", Advances in Cryptology-EUROCRYPT'88, Lecture Notes in Computer Science, vol. 330, Springer-Verlag, 1988, pp. 301-314.
[NES12]	NESSIE: "**New European Schemes for Signatures Integrity and Encryption**". See http://www.cryptonessie.org. Accessed on 21 Apr, 2012

[NIS12]	NIST: National Institute of Standards and Technology, 2012. http://csrc.nist.gov/groups/ST/hash/policy.html (accessed august 05, 2012)
[NIST12]	National Institute of Standards and Technology, Accessed on 01-Aug-2012 http://csrc.nist.gov/groups/ST/toolkit/rng/index.html,
[OZG06]	MeltemDoğanerÖzgan: A NEW NONLINEAR COMBINATION GENERATOR "MYBOUN", Master thesis, submitted to Graduate program in Electrical and Electronics Engineering, Bogazici University, 2006
[PP04]	Souradyuti Paul and Bart Preneel, "**A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher.**", Proceedings of Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, Lecture Notes in Computer Science, Vol. 3017, pp. 245{259, Springer-Verlag, Berlin, 2004.
[PP06]	S. Paul, B. Preneel "**On the (In) security of Stream Ciphers Based on Arrays and Modular Addition,**"Asiacrypt 2006 (X. Lai and K. Chen, eds.), vol. 4284 of LNCS, pp. 69-83, Springer-Verlag, 2006.
[PP10]	ChristofPaar and Jan Pelzl. "**Understanding Cryptography: A textbook for students and practitioners**", 2010 Springer p.7 ISBN 978-3-642-04100-6
[PPS05]	S. Paul, B. Preneel, and G. Sekar "**Distinguishing Attacks on the Stream Cipher Py**" available at http://www.ecrypt.eu.org/stream/papersdir/2005/081.pdf
[PPS06]	S. Paul, B. Preneel, G. Sekar, "**Distinguishing Attacks on the Stream Cipher Py**," Fast Software Encryption-FSE 2006 (M. Robshaw, ed.), vol. 4047 of LNCS, pp. 405-421, Springer Verlag, 2006.
[PRE09]	Bart Preneel"**Understanding Cryptography**", Springer. 2009
[QS01]	Jean–Jacques Quisquater and David Samyde. "**Electro Magnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards**". In Smart Card Programming and Security (E-smart 2001), Cannes, France, LNCS 2140, pp.200-210, September 2001.
[RAN13]	Random.org. Available at http://www.random.org/randomness/ Accessed on 26-Feb, 2013
[RM08]	Robshaw, Matthew. "**The eSTREAM project.**" New Stream Cipher Designs. Springer Berlin Heidelberg, 2008. 1-6.
[ROB95]	Robshaw, M. "**Block Ciphers.RSA Laboratories Technical Report TR-601"**, August 1995.Available at: http://www.rsasecurity.com/rsalabs
[ROS00]	Greg Rose "**S16 &S32: Fast Stream Cipher based on Linear Feedback over GF (2n)**". 2000 Available at http://www.home.aone.net.au/qualcomm

[ROS198]	G. Rose "**A Stream Cipher based on Linear Feedback over GF(2^8)**" in C. Boyd Editor Proc. Australian Conference on Information Security and Privacy SpringerVerlag 1998.
[ROS298]	G. Rose "**S32: A Fast Stream Cipher based on Linear Feedback over GF(232)**".Unpublished report QUALCOMM Australia Suite 410 Birkenhead Point Drummoyne NSW 2047 Australia 1998. Available at http: www.home.aone.net.au qualcomm.
[ROS398]	G. Rose "**SOBER II: A Fast Stream Cipher based on Linear Feedback over GF(232)**".Unpublished report QUALCOMM Australia Suite 410 Birkenhead Point Drummoyne NSW 2047 Australia 1998. See http: www.home.aone.net.au qualcomm
[ROS498]	G. Rose "**SOBER: A Stream Cipher based on Linear Feedback over GF (28)**". Unpublished report QUALCOMM Australia Suite 410 Birkenhead Point Drummoyne NSW 2047 Australia 1998.See http: people.qualcomm.com ggr.QC
[ROS598]	G. Rose and P. Hawkes "**The t-class of SOBER stream ciphers**" unpublished report QUALCOM Australia 1998. See http: www.home.aone.net.au qualcomm.
[SCH96]	Bruce Schneider: **Applied Cryptography**, Second Edition: Protocols, Algorithms, and Source Code in C. (1996)
[SHA49]	Shannon, Claude E. "**Communication theory of secrecy systems.**" Bell system technical journal 28.4 (1949): 656-715.
[SHI13]	Shirey, R., "**Internet Security Glossary**", Request for Comments 28, May 2000. http://www.ietf.org/rfc/rfc2828.txt (accessed January 02, 2013)
[SIE85]	T. Siegenthaler, "**Decrypting a class of stream ciphers using ciphertext only**," IEEE Trans. Computers, vol. C-34, no. 1, pp. 81–84, 1985.
[SIE86]	T. Siegenthaler, "**Design of Combiners to Prevent Divide and Conquer Attacks**", Advances in Cryptology-CRYPTO'85, H. C. Williams (Ed.), LNCS 218, Springer-Verlag, 1986, pp. 273-279
[SIM00]	Simon Singh. "**The Code Book: The Science of secrecy from Ancient Egypt to Quantum Cryptography**". Anchor Books, New York, 2000 ISBN 0-385-49532-3
[SPP071]	G. Sekar, S. Paul, B. Preneel, "**Related-key Attacks on the Py-family of Ciphers and an Approach to Repair the Weaknesses**," Indocrypt 2007
[SPP072]	GauthamSekar, Souradyuti Paul and Bart Preneel. "**Related-key Attacks on the Py-family of Ciphers and an Approach to Repair the Weaknesses**". Progress in Cryptology-INDOCRYPT 2007. Springer Berlin Heidelberg,2007 58-72

[SPP073]	GauthamSekar, Souradyuti Paul, and Bart Preneel **"Attacks on the Stream Ciphers TPy6 and Py6 and Design of New Ciphers TPy6-A and TPy6-B"** available at http://eprint.iacr.org/2007/436.pdf
[SPP074]	GauthamSekar, Souradyuti Paul, and Bart Preneel **"New Weaknesses in the Keystream Generation Algorithms of the Stream Ciphers TPy and Py"** available at http://eprint.iacr.org/2007/230.pdf 2, 3,11
[SPP075]	GauthamSekar, Souradyuti Paul, and Bart Preneel, **"Weaknesses in the Pseudorandom Bit Generation Algorithms of the Stream Ciphers TPypy and TPy"**, available at http://eprint.iacr.org/2007/075.pdf 2,3,11
[ST13]	Adi Shamir and EranTromer. **"Acoustic cryptanalysis: on nosy people and noisy machines"**. Available from: http://www.wisdom.weizmann.ac.il/~tromer/acoustic. Accessed on 11 Jan. 2013
[SX10]	Standaert, Francois-Xavier.**"Introduction to side-channel attacks"**. In: Verbauwhede, I.M.R. (ed.) Secure Integrated Circuits and Systems, pp. 27–42. Springer, Heidelberg (2010) ISBN: 978-0-387-71827-9
[TK12]	P. Topark and P. Kanivichaporn **"SOBER-t-16 and SOBER-t-32"** Available at http://citeseerx.ist.psu.edu/viewdoc. Accessed on 11 Mar. 2012
[VOR07]	Martin Voros" **Algebraic Attack on Stream Ciphers"**, Master's thesis, submitted to COMENIUS UNIVERSITY, Department of Computer Science. 2007
[WAT04]	D.Watanabe, A. Biryukov, and C. De Canniere. **"A Distinguishing Attack of SNOW 2.0 with Linear Masking Method"**. In Selected Areas in Cryptography (SAC 2003), LNCS 3006, pp. 222{233, Springer-Verlag, 2004
[WF04]	Dai Watanable and soichiFuruya **"A MAC forgery attack on SOBER-128"** Proc. Fast Software Encryption 2004 Springer 2004
[WIN08]	Myat Su Mon Win" **A New Approach to Feedback Shift Register"** World Academy of Science, Engineering and Technology 48 2008 pp. 185—189
[WP06]	H. Wu and B. Preneel **"Key Recovery Attack on Py and Pypy with Chosen IVs."** available at http://www.ecrypt.eu.org/stream/papersdir/2006/052.pdf
[WP07]	H.Wu and B.Preneel **"Differential Cryptanalysis of Stream Ciphers Py, Py6 and Pypy"**, Eurocrypt 2007 (MoniNaored) Vol. 4515 of LNCS, pp 276-290, Springer-Verlag 2007. 3,11
[ZF05]	Y. Zhou and D. Feng, **"Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing"**. NIST Physical Security Testing Workshop, Hawaii, USA, Sep. 2005. Cryptology ePrint Archive, Report 2005/388, 2005, http://eprint.iacr.org

[ZF09]	B. Zhang and D. Feng, "**An Improved Fast Correlation Attack on Stream Ciphers**", SAC'08, to appear, 2009.
[ZJZ09]	Wang Zhiyaun Huang Jianhua and Guan Ziming "**The SOBER Family Ciphers Reconfigurable Processing Arrhitecture Design** " fifth internation conferenee on Information assurance Seeurity 2009
[ZYW91]	KenchengZeng, Chung-Hung Yang, Dah-Yea Wei and T.R.N Rao."**Pseudorandom Bit Generators in Stream-Cipher Cryptography**" IEEE (1991)

Publications

Appendix I

[1] Bokhari, M. U.and Faheem Masoodi. "**BOKHARI: A new software oriented stream cipher: A proposal**." Information and Communication Technologies (WICT), 2012 World Congress on. IEEE, 2012.

[2] Bokhari, M. U., Shadab Alam, and Faheem Syeed Masoodi. "**Cryptanalysis Techniques for Stream Cipher: A Survey**." International Journal of Computer Applications 60.9 (2012).

[3] Faheem Masoodi, Shadab Alam and M. U. Bokhari, "**An Analysis of LFSR used in Stream Ciphers**", International Journal of Computer Applications, USA. Volume 46, No 17.

[4] Bokahri, Shadab and Faheem. "**A Review of Py (Roo) Stream Cipher and its Variants**". In Proceedings of the 5th National Conference; INDIACom-2011 Computing For Nation Development, March 10 – 11, 2011 Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi

[5] Faheem Masoodi, Shadab Alam and M U Bokhari **"SOBER Family of Stream Ciphers: A Review"**: International Journal of Computer Applications (2011), Vol. 23.

[6] Bokhari M. U. and Faheem Masoodi, "**Comparative Analysis of Structures and Attacks on Various Stream Cipher**", Proceedings of the 4th National Conference; INDIACom-2010. pp. 236—238.

Source Code of BOKHARI Stream Cipher

Appendix II

```
/*@Bokhari.c source code in gcc compiler.
  @Faheem Masoodi
*/

#define N 17
#define CONST_C 0x6996
#define WORDSIZE 16
#define WORD unsigned short

/* multiplication tables */
#include "multab.h"
#include "sbox.h"

WORD Initial[N];    /* Initial contents of shift register -- key
schedule */
WORD R[2*N];        /* Working storage for the shift register */
WORD konst;         /* key dependent non-linear function */
WORD *rp;           /* current offset in sliding window buffer */

/* external interface declarations */
void bokhari_key(unsigned char key[], int keylen);
void bokhari_genbytes(unsigned long frame, unsigned char buf[],
int nbytes);

/*
 * FOLD is how many register cycles need to be performed after
combining the
 * last byte of key and non-linear feedback.
 */
#define FOLD N          /* how many iterations of folding to do */
#define KEYP 15         /* where to insert key words */
#define FOLDP 4         /* where to insert non-linear feedback */
#define INITKONST 0x6996
/* end of bokhari header */


#define XOR_LAYER(d1, s1, d2, s2) \
    d1 ^= s1; \
    d2 ^= s2; ;


#define ADD_LAYER(d1, s1, d2, s2) \
    d1 += s1; \
    d2 += s2;


#define SBOX_LAYER(G, d1, s1, d2, s2) \
    d1 ^= G1(s1); \
    d2 ^= G2(s2);

#define bokhari_UPDATE(a, b, c, d) \
    XOR_LAYER(b, c, d, a) \
```

```
    ADD_LAYER(a, d, c, b) \
    SBOX_LAYER(G1, d, c, b,a) \
    SBOX_LAYER(G2, a, d, c d,) \
    ADD_LAYER( b, c, d, a) \
    XOR_LAYER(a, c, c, b)


/* cycle the contents of the shift register */
#define cycle(R, rp) \
{ \
    *rp = rp[N] = MUL0xE382(rp[15]) ^ rp[4] ^ MUL0x673C(*rp); \
    if (++rp == &R[N]) \
        rp = R; \
}

/* Return a non-linear function of some parts of the register.
 * The positions of the state bytes form a maximal span full
positive
 * difference set, and are 0, 1, 6, 13, 16.
 */

#define nltap(R, rp) \
    ((nltap_t = *rp + rp[16]), \
    (((SBox[nltap_t >> 8] ^ (nltap_t & 0xFF)) + rp[1] + rp[6]) ^
konst) + rp[13])

/* Definition of G function*/
#define G1(x) \
    sbox[x & 0xFF] ^ \
    sbox[(x >> 8) & 0xFF]

#define G2(x) \
    sbox[x & 0xFF] ^ \
    sbox[(x >> 8) & 0xFF]

#define G3(x) \
    sbox[x & 0xFF] ^ \
    sbox[(x >> 8) & 0xFF]

#define G4(x) \
    sbox[x & 0xFF] ^ \
    sbox[(x >> 8) & 0xFF]


/* definition of function f1. the specified four arguments are
0,4,15 and 17 where 17=0^4^15 */

#define bokh_nlfsr_update(R, rp)\
(nlfsr= *rp ^ rp[4] ^ rp [15] ^ \
bokhari_UPDATE ( *rp , rp[4] , rp [15] , (*rp ^ rp[4] ^ rp [15]
)))


/* load some key material into the register */
```

```
static void
loadkey(unsigned char key[], int keylen)
{
    register WORD     nltap_t;
    int i;

    /* start folding in key, odd byte first if there is one */
    if ((keylen & 1) != 0)
    {
     if (&rp[KEYP-N] < R)
         rp[KEYP] += key[0];
     else
         rp[KEYP-N] = rp[KEYP] = rp[KEYP] + key[0];
        cycle(R, rp);
     if (&rp[FOLDP-N] < R)
         rp[FOLDP] ^= nltap(R, rp);
     else
         rp[FOLDP-N] = rp[FOLDP] = rp[FOLDP] ^ nltap(R, rp);
    }
    for (i = keylen & 1; i < keylen; i += 2)
    {
     if (&rp[KEYP-N] < R)
         rp[KEYP] += (key[i] << 8) + key[i+1];
     else
         rp[KEYP-N] = rp[KEYP] = rp[KEYP] + (key[i] << 8) + key
[i+1];
        cycle(R, rp);
     if (&rp[FOLDP-N] < R)
         rp[FOLDP] ^= nltap(R, rp);
     else
         rp[FOLDP-N] = rp[FOLDP] = rp[FOLDP] ^ nltap(R, rp);
    }

    /* also fold in the length of the key */
    if (&rp[KEYP-N] < R)
     rp[KEYP] += keylen;
    else
     rp[KEYP-N] = rp[KEYP] = rp[KEYP] + keylen;

    /* now diffuse */
    for (i = 0; i < FOLD; ++i)
    {
     cycle(R, rp);
     if (&rp[FOLDP-N] < R)
         rp[FOLDP] ^= nltap(R, rp);
     else
         rp[FOLDP-N] = rp[FOLDP] = rp[FOLDP] ^ nltap(R, rp);
    }
}

/* calculate initial contents of the shift register */

void
```

```
bokhari_key(unsigned char key[], int keylen)
{
    register WORD     nltap_t;
    int i;

    /* fill the register with fibonacci numbers */
    R[0] = R[1] = 1;
    for (i = 2; i < N; i++)
        R[i] = R[i-1] + R[i-2];

    /* initialise the pointers and start folding in key */
    rp = R;
    konst = 0;
    loadkey(key, keylen);

    /* save state and key word for nonlinear function */
    cycle(R, rp);
    konst = nltap(R, rp);
    for (i = 0; i < N; i++)
        Initial[i] = rp[i];

}

/* set KONST to new value

static void
bok_genkonst()
{
    WORD   newkonst; // provide cycle uts arguments

    do {
      cycle(c->R);
      newkonst = nltap(c);
    } while ((newkonst & 0xFF000000) == 0);
    konst = newkonst;
}

/* Fold in the per-frame key */

void
bokhari_seckey(unsigned char seckey[], int seckeylength)
{
    register int    i;

    /* copy initial contents */
    for (i = 0; i < N; i++)
        R[i] = Initial[i];
    rp = R;

    loadkey(seckey, seckeylength);


}
```

```
/* XOR pseudo-random bytes into buffer */

#define XORWORD(p,v) (p[0] ^= ((v) >> 8), p[1] ^= (v), p += 2)
void
bokhari_gen(unsigned char *buf, int nbytes)
{
    unsigned char   *endbuf;
    register WORD     t = 0;
    register WORD     nltap_t;

    /* assert((nbytes & 1) == 0) */
    endbuf = &buf[nbytes];
    while (buf < endbuf)
    {

        cycle(R, rp);

      XORWORD(buf, t);
    }
}

/* encrypt/decrypt a frame of data */

void
bokhari_genbytes(unsigned long frame, unsigned char *buf, int
nbytes)
{
    unsigned char   framebuf[4];

    framebuf[0] = (frame >> 24) & 0xFF;
    framebuf[1] = (frame >> 16) & 0xFF;
    framebuf[2] = (frame >>  8) & 0xFF;
    framebuf[3] = (frame) & 0xFF;
    bokhari_seckey(framebuf, 4);
    bokhari_gen(buf, nbytes);
}

#ifdef TEST
#include <stdio.h>
#include <string.h>
#include "hexlib.h"

/* test vectors */
typedef unsigned char   uchar;
typedef unsigned long   word32;
uchar   *testkey = (uchar *)"test key 128bits";
word32  testframe = 1L;
uchar   testbuf[20];
WORD testkonst = 0x4b09;
char    *testout =
    "36 c3 df aa d1 63 41 46 01 34 05 13 8f 69 da 5e 21 45 44
fe";
char    *zeros =
    "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00";
#define ITERATIONS 999999
char    *iterout =
    "11 31 c2 5c d6 37 58 8f 6f b1 77 65 04 2d f4 69 47 f2 e5
70";
uchar   bigbuf[1024*1024];

void
test_bokhari(void)
{
    long   i;

    /* test encryption */
    bokhari_key(testkey, strlen((char *)testkey));
    printf("%14s: %04x\n", "konst", konst);
    if (konst != testkonst)
      printf("Expected %04x, got %04x.\n", testkonst, konst);
    bokhari_genbytes(testframe, testbuf, sizeof testbuf);
    hexprint("testbuf", testbuf, sizeof testbuf);
    hexcheck(testbuf, testout, sizeof testbuf);
    /* test decryption */
    bokhari_key(testkey, strlen((char *)testkey));
    bokhari_genbytes(testframe, testbuf, sizeof testbuf);
    hexprint("decrypted", testbuf, sizeof testbuf);
    hexcheck(testbuf, zeros, sizeof testbuf);
    /* test iteration */
    bokhari_key(testkey, strlen((char *)testkey));
    bokhari_genbytes(testframe, testbuf, sizeof testbuf);
    for (i = 0; i < ITERATIONS; ++i) {
      bokhari_key(testbuf, 16);
      bokhari_gen(testbuf, sizeof testbuf);
    }
    hexprint("iterated", testbuf, sizeof testbuf);
    hexcheck(testbuf, iterout, sizeof testbuf);
}

void
time_bokhari(void)
{
    word32      i;

    bokhari_key(testkey, strlen((char *)testkey));
    for (i = 0; i < 100; ++i)
        bokhari_genbytes(i, bigbuf, sizeof bigbuf);
}

void
lsb_bokhari(void)
{
    register int i;

    bokhari_key(testkey, 8);
    for (i = 0; i < 1024; ++i)
      cycle(R, rp);
```

```
    for (i = 0; i < 1024; ++i) {
      cycle(R, rp);
      printf("%lx", (unsigned long)(*rp & 0x1));
    }
    printf("\n");
}

#ifndef KEYLOAD
int
main(int ac, char **av)
{
    int         n;
    uchar       key[16];
    int         keysz;
    word32      hook;

    if (ac == 2 && strcmp(av[1], "-test") == 0) {
        test_bokhari();
        return nerrors;
    }
    if (ac == 2 && strcmp(av[1], "-time") == 0) {
        time_bokhari();
        return 0;
    }
    if (ac == 2 && strcmp(av[1], "-lsb") == 0) {
        lsb_bokhari();
        return 0;
    }

    if (ac >= 2)
        hexread(key, av[1], keysz = strlen(av[1]) / 2);
    else
        hexread(key, "0000000000000000", keysz = 8);
    sscanf(ac >= 3 ? av[2] : "00000000", "%lx", &hook);
    sscanf(ac >= 4 ? av[3] : "10000", "%d", &n);

    bokhari_key(key, keysz);
    if (n > sizeof bigbuf) n = sizeof bigbuf;
    bokhari_genbytes(hook, bigbuf, n);
    hexbulk(bigbuf, n);
    return 0;
}
#else
#include "stdlib.h"
main(int ac, char **av) {
    int         i, j;
    int         n;
    char    c;
    WORD    state1, state2;
    uchar       key[16];

    sscanf(ac > 1 ? av[1] : "0^", "%x%c", &n, &c);
    for (i = 0; i < 2500; ++i) {
      /* random key */
```

```
        for (j = 0; j < sizeof key; ++j)
            key[j] = random() & 0xFF;
        bokhari_key(key, sizeof key);
        bokhari_seckey("\0\0\0\0", 4);
        state1 = R[(r+n)%N];
        bokhari_seckey("\0\0\0\1", 4);
        state2 = R[(r+n)%N];
        printf("0x%04x\n",
            c == '^' ?
              state1 ^ state2
              :
              (state1 - state2) & 0xFFFF);
    }
}
#endif
#endif
```